Основи на JavaScript

II-ра част

10.11.2020

Владислав Илиев
vladislav.iliev@sap.com

Public

Съдържание

- ✓ Алгоритъм за конвертиране
- ✓ Масиви
- ✓ Обекти
- ✓ Функции

Преговор

Кои са типовете данни в JavaScript?

- ✓ boolean, null, undefined, number, string, Object , BigInt, *symbol

Кои са "falsy" стойностите в JavaScript?

- ✓ false, 0, undefined, null, "" (празен низ), NaN

Математика?

```
(5 / 0) == Infinity
```

```
(0.1 + 0.2) != 0.3
```

```
(0.1 + 0.2) == 0.30000000000000004
```

Ако се нуждаете от прецизност при изчисления на числа с плаваща запетая, използвайте библиотеки като mathjs

Алгоритъм за конвертиране

Алгоритъм за конвертиране

Въведение

```
let name = "Haralampi";
let year = "1998"; // string
```

```
if (year) {
        alert("thanks for entering an year");
} else {
        alert("you must enter a year");
}
```

```
if (year == 2000) {
        alert("you win 1 million euro!");
}
```

```
let currentYear = 2018; // int
let age = currentYear - year; // string - int = int
let helloString = "Hello," + name + ", aged " + age; // string + int
alert(helloString);
```

```
if (age < 18 || age < 21 && isUSA()) {
        alert("you should not …");
}
```

Алгоритъм за конвертиране
Въведение

```
1 if ("potato") {
2   console.log("potato" == false);
3 }
```

?

false

```
1 if ("potato") {
2   console.log("potato" == true);
3 }
```

?

false

ЗАЩО?

"Coercion/Abstract Equality Comparison Algorithm"

(Алгоритъм за конвертиране)

Алгоритъм за конвертиране

Описание оператор за сравняване == (1/5)

X == Y ?

Тип на X	Тип на Y	Резултат
еднакви типове		погледнете оператора "==="
null	undefined	true
undefined	null	true
number	string	x == toNumber(y)
string	number	toNumber(x) == y
boolean	(any)	toNumber(x) == y
(any)	Boolean	x == toNumber(y)
string or number	Object	x == toPrimitive(y)
Object	string or number	toPrimitive(x) == y
Във всички останали случаи…		false

undefined е равен на null. Всичко друго се конвертира почти винаги до число преди да се извърши сравнение.

Алгоритъм за конвертиране
Описание оператор за сравняване == (2/5)

toNumber(Z)

Тип на Z	Резултат	Коментар
undefined	NaN	Number(Z)
null	+0	
boolean	**1** при **true**, **0** при **false**	
number	Z (няма конверсия)	
tring (parseFloat)	• toNumber("777") или toNumber("777FMI") -> 777 • toNumber("FMI") или toNumber("FMI777") -> NaN	parseFloat(Z)
Object	toNumber(toPrimitive(Z))	

Алгоритъм за конвертиране
Описание оператор за сравняване == (3/5)

toPrimitive(Z)

Тип на Z	Резултат
Object	1. valueOf(Z) връща примитив или 2. toString(Z) В противен случай – хвърля се грешка
Други	Връща Z

Алгоритъм за конвертиране

Описание оператор за сравняване === (4/5)

X === Y ?

Тип X и Y	Стойности	Резултат
Типа на X се различава от типа на Y		false
undefined	undefined	true
null	null	true
number	x е със същата стойност като y (но не и NaN)	true
string	x и y са идентични символи	true
boolean	x и y са и двете true или и двете false	true
Object	x и y реферират един и същи обекти	true
В противен случай…		false

Алгоритъм за конвертиране
Събиране и изваждане с низове (5/5)

X + Y = ?, X - Y = ?

Израз	*Резултат*
"hello" + "world"	"helloworld"
"5" + "5"	"55"
"5" + 5	"55"
5 + true	6
"5" - "5"	0
"5" - 5	0

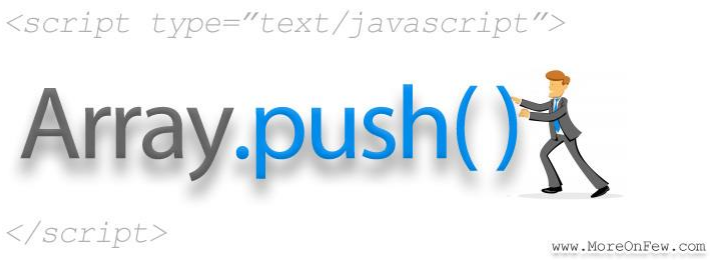
Substract operator rules: here

Demo

Public

Масиви

- ✓ Деклариране и инициализация
- ✓ Достъпване на елементи
- ✓ Динамичност
- ✓ Сортиране на масиви



Масиви

Преглед

- ✓ **Деклариране и инициализация**
  - ✓ new Array(elements)
  - ✓ new Array(length)
  - ✓ var array = […]

```
1 let myArray = new Array("Pesho", "Gosho", "Ivan");
2
3 let myArray = new Array(3); // array with 3 empty slots
4
5 let myArray = ["Pesho", "Gosho", "Ivan"]; // recommended
```

Масиви

Преглед

- ✓ **Достъпване на елементи**

```
1 let myArray = ["HTML", "CSS"];
2 console.log(myArray[0]); // print "HTML"
3 console.log(myArray[1]); // print "CSS"
4 console.log(myArray["1"]); // print "CSS"
5 console.log(myArray.0); // error
```

Масиви

Преглед

- ✓ **Динамичност**
  - ✓ Добавяне на последен елемент – Array.push()
  - ✓ Премахване на последен елемент – Array.pop()
  - ✓ Вмъкване на пръв елемент – Array.unshift()
  - ✓ Премахване на пръв елемент – Array.shift()

```
1 let myArray = ["Node.js", "React", "AngularJS"];
2 console.log(myArray.pop()); // print "AngularJS"
3 console.log(myArray); // ["Node.js", "React"]
4
5 console.log(myArray.push("Vue.js"));// print 3(length)
6 console.log(myArray); // ["Node.js", "React", "Vue.js"];
```

Масиви

Методи

- ✓ **Сортиране**
  - ✓ Array.sort()

```
let myArray = ["C", "A", "D", "B", 80, "E“, 9];

myArray.sort();

console.log(myArray); // [80, 9, "A", "B", "C", "D", "E"]
```

  - ✓ Array.sort(<compareFunction>)

```
myArray.sort((a,b)=> {
    console.log(a, isNaN(a), b, isNaN(b));
    if (!isNaN(a) && !isNaN(b)) return a - b;
 });

```

Масиви

Методи

- **Методи за обхождане**
  - array.forEach(function(item, index){})
    - Итерира по елементите на масива
  - array.filter(function(item, index){})
    - Връща нов масив с елементи, които удовлетворяват условието
  - array.map(function(item, index){})
    - Връща нов масив, попълнен с резултатите от функцията за всеки item

```
1 let myArray = ["A', "B", "C", "D"];
2
3 myArray.forEach(function(item, index) {
4       console.log(item);
5       console.log(index);
6 });
```

Масиви

Методи

- **Методи за "отрязване" и "заменяне"**
  - array.slice(begin, end)
    - Връща отрязък от масива
  - array.splice(start, deleteCount, item1, item2, ...)
    - Подменя част от елементите с нови елементи

```
1 let myArray = ["A", "B", "C", "D"];
2 console.log(myArray.slice(1, 3)); // ?
3 console.log(myArray.splice(0, 3, "E", "F")); // ?
```

Масиви

Методи

- **Други полезни методи при масиви**
  - array.reverse()
    - Обръща реда на елементите в масива
  - array.concat(elements)
    - Добавя подадените елементи към края на масив и връща нов масив
  - array.join(separator)
    - Връща нов низ с елементите на масива разделени със стойността на *separator*
  - array.indexOf(element)
    - Връща индекса на първия елемент, който е равен на element или -1 ако не е намерен такъв
  - array.lastIndexOf(element)
    - Връща последният елемент, който е равен на element

```
1 let myArray = ["A", "B", "C", "D"];
2
3 console.log(myArray.indexOf("B")); // print 1
```

Demo

Public

Обекти

- Какво са обектите?
- Обектите в JavaScript
- **J**ava**S**cript **O**bject **N**otation (JSON)
- Асоциативни масиви
  - речници и мапове в JavaScript

Обекти

Overview

- **Какво са обектите?**
  - Абстракция, която симулира реални обекти от живота
  - Обектите както в реалният живот, така и в програмирането имат състояния и поведение
  - Създаден обект от определен тип се нарича **инстанция**

- **Обекти в JavaScript**
  - В JavaScript почти всичко е обект
  - Стойностите в обектите се представят като асоциация име и стойност
  - Създават се по няколко начина

Обекти

Overview

✓ Създаване и използване

```
let myObject = {};
let myObject2 = new Object();
let myObject3 = Object.create(...);
```

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

```
console.log(person.firstName);
console.log(person.lastName);
console.log(person.sayHello());
```

Обекти

JSON обекти

- Какво е JSON
  - **J**ava**S**cript **O**bject **N**otation
  - Стандартен начин за дефиниране на обекти
  - Често се използва за пренос на данни между сървър и клиент
  - Често данните в JSON формат представяват масив от обекти

```
{
    "formatted_address": "1600 Amphitheatre Parkway, …, USA",
    "geometry": {
        "location": {
            "lat": 37.4224764,
            "lng": -122.0842499
        },
        "location_type": "ROOFTOP"
    },
    "types": ["street_address"],
    "status": "OK"
}
```

Part of geocoding response from Google Geocoding Api. Lookup for address "1600 Amphitheatre Parkway"

Обекти

Асоциативни масиви

- ✓ В JavaScript обектите могат да се ползват и за асоциативни масиви

```
let colors = {
    "orange": "#FFA500",
    "black": "#000000",
    "white": "#FFFFFF",
    "maroon": "#800000",
    "navy": "#000080"
}
```

Обекти

Вградени обекти

- **Built-in JavaScript Objects**
  - Date
  - Arrays
  - RegExp
  - Promise

Demo

Public

Функции

- ✓ Деклариране и викане на функции
- ✓ Функции с аргументи
- ✓ Връщани стойности
- ✓ Scope на функция
- ✓ Overloading на функция

$$f(x, y) = x + y$$

Функции

Overview

- ✓ **Какво са функциите?**
  - ✓ Парчета от код
  - ✓ Решават даден проблем
  - ✓ Могат да бъдат викани
  - ✓ Могат да приемат параметри и да връщат стойност
- ✓ **Използване на функции**
  - ✓ Разделяне на проблема на малки парчета
  - ✓ Организация на кода
  - ✓ Подобряване на четимостта на кода
  - ✓ Избягване на повтаряне на код
  - ✓ Кодът става преизползваем

Функции

JavaScript функции

- ✓ **Какво са функциите в JavaScript**
  - ✓ Всяка функция има **име (освен анонимните, които нямат)**
    - ✓ Използва се за викане на функцията
    - ✓ Описва целта на функцията
  - ✓ Функциите в JavaScript **нямат тип**, който трябва да се върне, но могат да **връщат резултат**
  - ✓ Ако не сме казали на функцията какво да върне, тя връща **undefined**
  - ✓ Функциите в JavaScript могат да имат **параметри**
  - ✓ Функциите в JavaScript имат **тяло**
    - ✓ Тялото съдържа кода за изпълнение
    - ✓ Загражда се от { }

```
1 function printNumbers(n) {
2     for (var i = 1; i <= n; i++) {
3         console.log(i);
4     }
5 }
```

Функции

Деклариране и използване

- ✓ **Как се декларират функции в JavaScript?**
  - ✓ Чрез конструктор за деклариране на функция

```
1  var print = new Function('console.log("Hello")');
```

  - ✓ Чрез декларатор за функция

```
1  function print() { console.log('Hello') };
```

  - ✓ Чрез израз за функция

```
1  var print = function() { console.log('Hello') };
2
3  var print = function printFunc() { console.log('Hello') };
```

  - ✓ В ES6 чрез стрелка (fat arrow function/lambda expression). Обвързани са със скоупа в който са дефинирани.

```
var print = () => console.log('Hello');
```

Функции

Извикване на функции и функции с параметри

- ✓ **Как се извикват функции в JavaScript?**
  - ✓ Име на функцията
  - ✓ Параметри (ако функцията има такива)
  - ✓ Точка и запетая =)

```
1  function printNumber(n) {
2      for (var i = 1; i <= n; i++) {
3          console.log(i);
4      }
5  }
6
7  printNumber(10);
```

```
1
2
3
4
5
6
7
8
9
10
<· undefined
```

Какво ще направи тази функция и какво ще върне ?

Функции

Извикване на функции и функции с параметри

- ✓ Функциите могат да бъдат викани от други функции
- ✓ Или от самите себе си ... (рекурсия)

```
1  function printNumber(n) {
2      if (n == 0) {
3          return;
4      }
5      console.log(n);
6      printNumber(n - 1)
7  }
8
9  printNumber(10);
```

Внимавайте с рекурсията =)

Функции

Аргументи на функции и обектът `arguments`

- Аргументите на една функция могат да са много и от всякакъв тип
  - Number
  - Object
  - Array
  - etc..
  - Дори **Function**

```
1  function car(owner, color) {
2      console.log('Type of the owner parameter is:  ' + typeof(owner));
3      console.log('Type of the color parameter is: ' + typeof(color));
4
5      return owner.firstName + ' ' + owner.lastName +
6          ' drives ' + color + ' car.';
7  }
8
9  car({ firstName: 'Martin', lastName: 'Hristov' }, 'red');
```

Функции

Параметри – по стойност и по референция

```
1  let mySallary = 10000;
2  let myObject = {name: "Peter", age: 20};
3
4  function x (p1, p2) {
5       console.log("Началото: 1: " + p1 + "2: " + p2.name + ": " + p2.age);
6
7       p1 = 15000;
8       p2.age = 21;
9
10      console.log("Край: 1: " + p1 + "2: " + p2.name + ": " + p2.age);
11      p2 = null;
12 }
13 console.log("Начални стойности на променливите: ");
14 console.log("1: " + mySallary);
15 console.log("2: " + myObject.name + ": " + myObject.age);
16
17 x(mySallary, myObject);
18
19 console.log("Стойности на променливите след извикване на функция:");
29 console.log("1: " + mySallary);
21 console.log("2: " + myObject.name + ": " + myObject.age);
```

Функции

Обектът *arguments*

- ✓ Обектът **arguments**
  - ✓ Всяка функция има такъв обект
  - ✓ Съдържа списък от подадените аргументи
  - ✓ Няма нужда да се подава като аргумент
  - ✓ Function overloading

```
function introduceYourself() {
    console.log(arguments);

    return 'Hello I am ' + arguments[0] + ' ' + arguments[1] +
        ' and I am ' + arguments[2] + ' years old from ' + arguments[3];
}

introduceYourself('Martin', 'Hristov', 20, 'Sofia');
```

Функции

Обхват на функция и променливи 1/4

- ✓ Променливите имат обхват
  - ✓ Обхвата на една променлива показва къде може да се достъпи тя
  - ✓ Глобални и локални променливи
  - ✓ Променлива декларирана без ключовата дума **var** става **глобална**

```
1  function fnScope() {
2      var a = 5; //local
3      b = 15 // global
4
5      if (true) {
6          var c = 20 // lives even after if scope
7      }
8
9      console.log(a);
10     console.log(b);
11     console.log(c);
12 }
```

Функции

Обхват на функция и променливи (hoisting) 2/4

- Обхват на функция и променливи (hoisting)?
  - Декларирането на променливите се изпълнява преди изпълнението на кода
  - Декларирането на променлива където и да е в кода е същото като декларирането ѝ на горе на scope-а ѝ
  - Това може да доведе до случай, в който променливата е декларирана, но не и инициализирана

Функции

Обхват на функция и променливи (hoisting) 3/4

```
var myVar = "Global";
function someAwesomeFunction() {
  console.log(myVar); //prints "Global"
  console.log(awesomeVar);//undefined but no ReferenceError, since the var
exists
  var awesomeVar = "Local";
  console.log(awesomeVar); // prints "Local"
}
```

```
var myVar = "Global";
function someAwesomeFunction() {
  //prints "undefined" because the local var is hiding the global
  console.log(myVar);
  var myVar = "Local";
  console.log(myVar); // prints "Local"
}
```

Функции

Обхват на функция и променливи (hoisting) 4/4

```
1 isItHoisted();
2 definitionNotHoisted();
3
4 function isItHoisted() {
5     console.log("Yes!");
6 }
7
8 var definitionNotHoisted = function () {
9   console.log("Definition not hoisted!");
10};
```

Функции

Overloading на функция

- ✓ Какво е overloading на функция?
  - ✓ Няколко функции с едно и също име, но с променлив брой аргументи
  - ✓ Function overloading в JavaScript **няма**
  - ✓ Всяка следваща функция със същото име като някоя минала презаписва старата
  - ✓ Но ... както почти всичко друго което липсва в JavaScript така и Overloading – а може да се **постигне**

- ✓ Съществуват няколко вида *възможности за overloading*
  - ✓ Променлив брой аргументи
  - ✓ Променлив тип аргументи
  - ✓ Незадължителни аргументи

Функции от по-висок ред

- ✓ Какво е функция от по-висок ред?
  - ✓ Функция която може да приема като параметър друга функция
  - ✓ Функция която връща като резултат функция

```
// Higher order function (that returns another function as a result)
let times = function (x){
  return function (y){
    return x * y;
  }
}
```

Demo

Public

Благодаря Ви за вниманието!

Контакти:

Владислав Илиев
vladislav.Iliev@sap.com

Филип Сидеров
filip.siderov@sap.com

SAP Labs Bulgaria
София, бул. Цариградско Шосе 111B