JAVASCRIPT BASICS

Jakub Baierl & Šimon Lomič

JAVASCRIPT DEVELOPERS @ ACKEE

ackee

“JavaScript is the only language developers don't learn to use before using it.”

“JavaScript is the World's Most Misunderstood Programming Language”

Douglas Crockford

“Easy to learn, hard to master.”

Mattias Petter Johansson

WHAT IS JAVASCRIPT?

```
<script>
    console.log("Hello, World!")
</script>
```

BASICS

- dynamic typed
- scripting
- cross platform
- client-side
- server-side
- standards & versions

```
var x = "I am a string!"; //string
x = 1; //now that is a number!
```

BASIC ELEMENTS

- operators
  - unary, arithmetic, relational, equality, bitwise
  - operator precedence
- control flow
  - loops
    - for, for in, for of, while, do while, break, continue
  - conditionals
    - if, else, switch, ternary operator
  - exceptions
  - timers

```
var z = x + y;
var txt1 = "What a very ";
txt1 += "nice day”;
```

```
z = "Hello" + 5;
```

```
for (var i = 0; i < 10; i++)
{
    result = i === "1";
    console.log(result);
}
setTimeout(function()
{
  console.log(result);
}, 3000);
```

Is Javascript hard to learn?

- Many things “just work”
- Easy to learn from existing code
- Simple concepts, but it is difficult to fully understand their behavior

Is Javascript hard to learn?

- Some concepts can be confusing for beginners
  - prototypes, functions as first class objects, hoisting
- Everything can be done in thousands of different ways
  - No clear conventions and principles

Is Javascript hard to learn?

JavaScript is a polarizing language, said to be full of “good parts” and “bad parts”.

- **Good Parts:**
  - free syntax allows to implement various paradigms
  - first class functions
  - closures
- **Bad Parts:**
  - type coersion
  - *with, eval, void*
  - *this* confusion (5 ways to set)

JAVA VS JAVASCRIPT

"Java is to JavaScript as ham is to hamster."

Jeremy Keith

Java	Javascript
static typing	dynamic taping
strong type checking	weak type checking
object oriented - **class-based**	object oriented - **prototype-based**
block-based scoping	function-based scoping

JAVA IS TO JAVASCRIPT AS...

http://javascriptisnotjava.io/

hung is to hunger
stab is to stabbing
sa is to save
bi is to birth
poop is to poopsicle
yo is to yolk
berry is to raspberrypi
come is to comedian
math is to maths
ea is to earth
as is to asshole
chic is to chicago
hog is to hogwarts
ant is to antigravity
bass is to basstard
crack is to crackcocaine
desk is to desktop
ping is to pingpong
four is to fourchan
dim is to dimple

anal is to analogies
java is to javascript
fire is to firefly
cram is to crampon
java is to javascript
cam is to camel
chair is to electricchair
dog is to dogma
bell is to belligerent
deaf is to deafinitely
moth is to mother
tips is to tipsy
poll is to pollock
toy is to toyota
men is to menace
chain is to chainmail
jew is to jews
sun is to sunder
master is to masterbait
son is to sonnet

con is to controller
mug is to mugshot
ayy is to ayylmao
app is to apple
helm is to helmet
car is to cartwheel
bob is to bobcat
red is to reddit
anal is to analysis
drag is to dragons
ben is to bendingover
round is to roundhouse
cock is to cocktail
barn is to barney
dog is to dogged
me is to memes
rag is to ragnarok
action is to actionscript
cheese is to cheesees
cat is to catamaran

crack is to crackalackasmacka
fallout is to falloutboy
coffee is to coffeescript
water is to watermellon
jam is to jamaica
thanks is to thanksobama
cool is to coolcat
butt is to notbutts
java is to javascript
ham is to hamwallet
coal is to coalition
bar is to barber
jack is to jackallantern
card is to cardassian
poop is to poopydiaperslol
mountain is to mountaindew
hi is to hitler
honestly is to honestlythisisagoodsite
war is to warsaw
for is to forget

promise is to compromise
ham is to hamburger
tree is to treehouse
poke is to pokemon
hue is to huehuehue
meme is to memento
bus is to business
red is to redemption
poo is to tablespoon
poo is to pool
lob is to lobster
bur is to burbon
count is to country
path is to sociopath
aspirin is to aspiring
ham is to hammer
cock is to cockatoo
hell is to helles
car is to carpet

So yes, they are quite different.

Javascript's "gotchas"

- JS has some really confusing parts
  - `"" == 0 // true` BUT `"" == "0" // false`
  - `a = 0; b = -0; a === b // true`

    `1/a === 1/b // false`

Javascript's "gotchas"

WTFJS: http://wtfjs.com/

A Collection of JavaScript Gotchas:
http://www.codeproject.com/Articles/182416/A-Collection-of-JavaScript-Gotchas

JavaScript Garden: documentation about the most quirky parts of JS
http://bonsaiden.github.io/JavaScript-Garden/

Common JavaScript Errors: The List
http://www.javascriptgotchas.com/gotchas/common-javascript-errors-and-mistakes.html

Javascript History in a nutshell

- Originally developed in 10 days in May 1995 by Bendan Eich for Netscape
  - intended as simpler scripting language for websites, complementing Java
  - soon after Microsoft implement compatible dialect of JS called JScript
- in 1997 organization ECMA standardized ECMAScript language
- June 2011: ECMAScript 5.1, now has in average 98% compatibility in all modern browsers
- June 2015: ECMAScript 6 released, compatibility with modern browsers is unsatisfactory (e.g. IE 11 has 15%, while being used by circa 20% users)
- ES8 in progress

- JavaScript, JScript and ActionScripts are considered dialects of ECMAScript

TOOLS

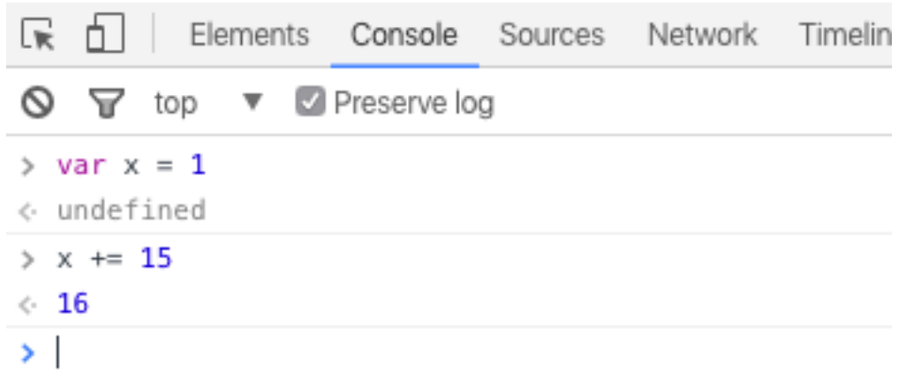
TOOLS > DEVELOPER TOOLS

- Google Chrome (as well as any other modern browser) provide set of powerful tools for debugging
  - How to open them:

Windows: **Ctrl** + **Shift** + **J**

Mac: **Cmd** + **Opt** + **I**

or Menu > Tools > Developer Tools



TOOLS > CONSOLE API

- JS's standard doesn't provide simple input/output
- we use custom ways on each runtime environment
- every modern ECMAScript runtime provides a **Console API** (although implementations can differ)**:**
  - `console.log()`
  - `console.warn()`
  - `console.error()`
  - `console.debug()`
  - `console.monitor()`
  - `console.monitorEvents()`
  - `console.time() console.timeEnd()`
  - `debugger;`

TOOLS > DOCUMENTATION

- Best JavaScript documentation is governed by Mozilla Developer Network :
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/ - every JS developer should have bookmarked
  - ECMAScript standards https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language_Resources

This is an experimental technology

Deprecated

Browser compatibility

Desktop | Mobile

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support	5	4.0 (2.0)	9	12	5

TOOLS > DOCUMENTATION

- Other useful links:
  - **Reference with fast search and offline support:** http://devdocs.io/
  - **Up-to-date browser support tables:** http://caniuse.com/
  - **Browser statistics:** http://www.w3schools.com/browsers/browsers_stats.asp
  - **Google JavaScript Style Guide:** https://google.github.io/styleguide/javascriptguide.xml

LANGUAGE ELEMENTS

TYPES OF VALUES

a value can be:

- **Primitive - raw value**
  - **number** - just any number, really (float, long, signed, …)
  - **string** - texts and characters
  - **boolean** - true or false, nothing else
  - "*nothing*" - ***undefined***, ***null***, we'll get to them
- **object** — *complex value,* anything else from *arrays* and *functions* to *custom types*

PRIMITIVES

NUMBERS · STRINGS · BOOLEAN · NULL/UNDEFINED

- simple “raw” value
- distinguishes from objects by not having *properties* nor *methods*

number

- unlike many other languages, JS doesn't distinguish between floating point numbers and integers

- every number is internally 64-bit Double -> approximately 16 decimal numbers big with maximum precision

9 999 999 999 999 999

number

- special values:
  - `Infinity, -Infinity` `1/0` `-1/0`
  - NaN *(not a number)* 0/0 Math.log(-3)
- safe and unsafe integers
  - double can represent bigger number than $9*10^{15}$ but not with integer precision
  - the integers inside interval ($-9*10^{15}$, $9*10^{15}$) are considered safe

string

- unlike in other languages, strings aren't arrays of characters, but standalone **IMMUTABLE** type
  - you can access characters at position, but you cannot change them

```
var str = "ahoj";
x = str[3]; // "a"  str[3]="b"; //no change
```

  - every operation (like concatenation) creates new string
    - every modern browsers implements them effectively

boolean

- true and false

boolean gotcha:

```
var x = new Boolean(false)
if (x) console.log("what?");
```

undefined and **null**

- **undefined**
  - primitive value, that is **automatically assigned** to variables, that have just been declared
  - means “I don’t know what’s inside this variable is”
- **null**
  - primitive value, that represents **intentional absence** of any object value
  - means “This variable has no value”

Operations on **primitives**

String

- charAt()
- indexOf
- substr()
- slice()
- split()
- toLowerCase(), toUpperCase

Number

- isInteger()
- isNan()
- toPrecision(),toFixed(), toExponential(), toString()
- isFinite()
- parseInt(), parseFloat()

Boolean

- toString()

But wait…
didn't we say primitives have no methods?

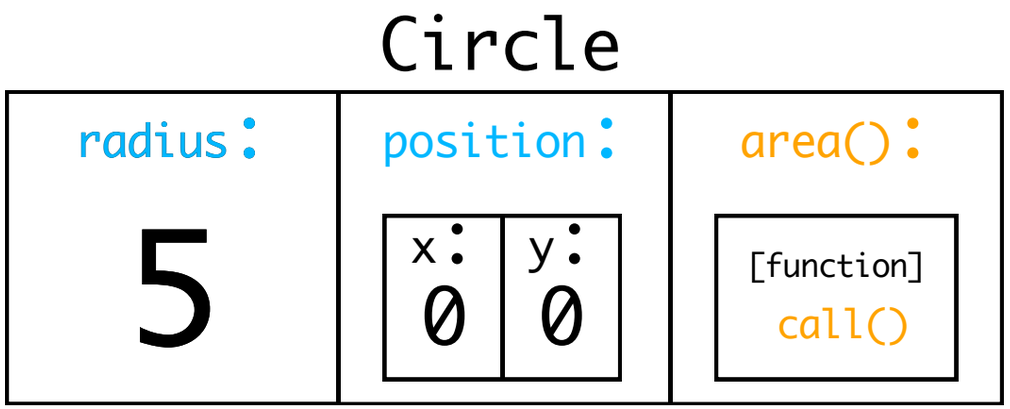
Primitive Wrapping Objects

- Each primitive has its own wrapping object
- Use them for explicit type coercion
- Are implicitly used for method calls on primitives

OBJECTS

- Basic building blocks of JS
- Set of **Name:Value** pairs
  - a.k.a - associative array, dictionary, hash, lookup table
  - Value can be either **properties** or **methods**

```
var Circle = {
    // properties:
    radius: 5,
    position: { x: 0, y: 0 },
    // methods:
    area: function() {
        return Math.PI * Math.pow(this.radius, 2);
    }
}
```



DATA TYPES: OBJECTS

- Creation
  - by object initializers

```
var obj = {};
```

  - by using a constructor function

```
var obj = new Object();
```

DATA TYPES: OBJECTS

- Accessing properties and methods
  - dot notation (for valid JS identifier - [A-Za-z_$0-9])

```
Circle.radius, Circle.area()
```

  - bracket notation

```
Circle["radius"], Circle["area"]()
```

    - The name of property / method must be string
      - non-strings are automatically casted using toString method

DATA TYPES: OBJECTS

- Objects are passed **by reference**
- Objects comparison is based on **identity**, not **equality**

More advanced tasks:

- Listing of all properties and methods of an object
- deep copy of objects

PRIMITIVES VS OBJECTS

- when assigning or passing into function
  - primitives are always **copied**
  - objects are **passed by reference**
- objects have properties and methods, primitives are simple “raw” value

ARRAYS

- Array is an object subtype, a.k.a. list, vector, …
- Array creating:
  - Array initializer: `var arr = [1,2,3];`
  - Array constructor: `var arr = new Array();`
- contains auto-updated property `.length`
- most used methods:

  `push, pop, indexOf, lastIndexOf, find, reverse, sort, splice, shift`

FUNCTIONS

- output arguments only as objects, implicitly return *undefined*
- `arguments` - implicit local variable within each function, = array of arguments
- Function is an **object** subtype, **length attribute** returns number of arguments
- Creation:
  - Function declaration
  - Function expression - anonymous functions
  - Named function expression

DATA TYPES: FUNCTIONS

- Functions are **First class citizens** - what's that?
  - allow operations: **passing** as argument, **returned** from functions, **assigned** to variable
- Functions as parameters of objects are called **methods**
  - using **this** keyword you can access the *context object*, the function was called on

OPERATORS

- **Arithmetic Operators**

  `- * / % ++ --`

- **Assignment Operators**

  `= += -= *= /= %=`

- **Comparison Operators**

  `== === != !== > <`
  `>= <= ?`

- **Logical Operators**

  `&& || !`

- **Assignment Operators**

  `= += -= *= /= %=`

DATA TYPES > OPERATORS

- **Bitwise Operators**

  `& | ^ << >> >>> ~`

- **String Concatenation Operators:**

  `+ +=`

- **Comma operator:**

  `,`

- **Ternary operator:**

  `? :`

- **Relational Operators**

  `in, instanceof`

- **Unary Operators**

  `void, delete, new, typeof`

TYPES OF VARIABLES

JS is **dynamically typed**
But, what does that mean?

Purpose of data types:

- Defines format and length of data associated to a **variable**
- Define operations I can do with that **variable**

DATA TYPES

JS assigns type to **values** instead of **variables**

- variable can contain value of any type

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

Can I just assume function will be called only with numbers?

```
// works only on numbers!!!
function sum(a, b) { return a + b; }
```

No. But I should document it or handle all possible types.

DATA TYPES

```
function sum(a, b) {
    a = parseFloat(a); b = parseFloat(b);
    if (isFinite(a) && isFinite(b))
        return a + b;
    else
        throw new Error("Invalid arguments");
}
```

Javascript is **"Duck typed"** language:

(runtime explicit type checking)

"When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck."

DATA TYPES

```
 1 var duck = {
 2     appearance: "feathers",
 3     quack: function() {
 4         return "Quack-quack!";
 5     }
 7 };
 8
 9 var someAnimal = {
10     appearance: "feathers",
11     quack: function() {
12         return "Whoof-whoof!";
13     }
15 };
16
17 function check(who) {
18     if ((who.appearance === "feathers") && (who.quack() === "Quack-quack!")) {
19         who.quack("I look like a duck!\n");
20         return true;
21     }
22     return false;
23 }
24
25 check(duck);  // true
26 check(someAnimal);  // true
```

DATA TYPES

Type checking problem in JS actually handled more complexly by these two mechanisms:

Type Coercion

JS implicitly casts types of primitives so that they can make work done

```
1 - true - "3" + 4
// 1
```

Introspection

programmer explicitly asks if value conforms to a given type

```
elephant instanceof Animal
typeof str == "string"
```

TYPE COERCION

- why does this work?

```
"2" - 1 == 1 // true
[43] > ["42"] // true
"1" == true // true
```

- but…

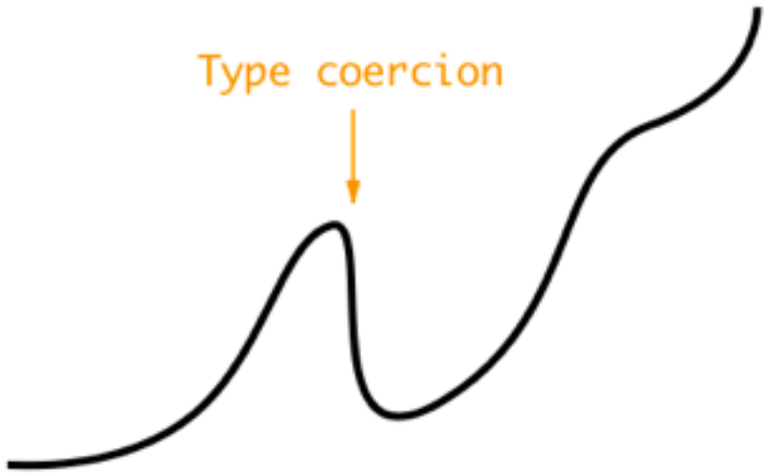
```
"2" + 1 == 1 // false
[43] > ["042"] // false
"2" == true // false
```

- Uh, oh!

DATA TYPES > TYPE COERCION

- **Type Coercion:**

useful feature
or
a flaw in the design of the language
(or somewhere in between!)



General opinion:
"coercion is magical, evil, confusing, and just downright a bad idea"

- **Type Coercion: Why do I have to learn this?**
  - happens on almost every line of code
    - `if (arr.length) {…}`
  - essential when parsing user input
  - deep understanding of language

DATA TYPES > TYPE COERCION

- **when coercion happens?**
  - when two types meat in single expression, or when particular type is expected
- **when that is?**
  - at operators and explicit casts
    - `if, while, do-while, &&, ||, +-*/%, | &<< >>, <,>,<=,>=, ==,!==`
  - **===, switch** -> “coercion shield”

DATA TYPES > TYPE COERCION

- **How does coercion coerce?**

  1. first it finds out which on type should operation happen

  2. then *converts* the arguments into this type

```
"44" - 2
    // operator - (a,b):
1) // a and b should be Numbers
2) return Number(“33”) - Number(2)
```

DATA TYPES > TYPE COERCION

1. **Find out which on type should operation happen**
   - on which types does the operator work?
   - what are the types of argument?

```
1 + true // 2
// strings always prevails
1 + "1" // “11”
// but expressions are evaluated from left
1 + 1 + "1" // “21”
```

2. **Convert the arguments into this type**

    1. To **Numbers**:

        - **Strings**: attempts parse string, otherwise NaN
        - **Booleans:** true: 1, false: 0
        - **Null:** 0
        - **Undefined**: NaN

2. **Convert the arguments into this type**

    2. To **Booleans**:

        - **Strings**: false from empty string, otherwise true
        - **Numbers:** false from: `0, Nan`, otherwise true
        - **Null:** false
        - **Undefined**: false

2. **Convert the arguments into this type**

3. To **Strings**:

- calls `toString()` method

```
{} + "2" // 2 ({}; +"2")
{} + {} // "[object Object][object Object]"
```

2. **Convert the arguments into this type**

What about **objects**?

- to **bool**: always *true*

```
var x = new Boolean(false)
if (x) console.log("what?");
```

- to other type:
  - firstly calls `toString()`
  - then converts to desired type

[*43*] > ["42"] // true, because

"[43]" > "[42]"

Coersion Tips:

Whenever you are not sure:

- use === and explicit coercion
- use JS console
- look into ECMAScript standard for the exact algorithm

TYPEOF OPERATOR

- Another way to solve type checking problem
- **typeof** operator asks what kind of type value is, and returns one of following strings:

“number”, “boolean”, “string”, “undefined”, “object”, “function”

Wait…
Where is **null**? And also, when there’s **function**, why there’s no **array**?

API

API

- containers
- math, time, date
- DOM

jQUERY

JQUERY

- changing DOM
- events
- jQUERY UI

ackee

JAKUB BAIERL
&
ŠIMON LOMIČ

@borecekbaji
jakub.baierl@ackee.cz