

THE COMPLETE JAVASCRIPT COURSE

FROM ZERO TO EXPERT!

SECTION JAVASCRIPT FUNDAMENTALS – PART 1

LECTURE BOOLEAN LOGIC

The JavaScript logo, consisting of a yellow square with the letters 'JS' in black.

BASIC BOOLEAN LOGIC: THE AND, OR & NOT OPERATORS

A AND B

*"Sarah has a driver's license
AND good vision"*

Possible values

		A	
B	AND	TRUE	FALSE
	TRUE	TRUE	FALSE
	FALSE	FALSE	FALSE

Results of operation, depending on 2 variables

true when **ALL** are true

No matter how many variables

A OR B

*"Sarah has a driver's license
OR good vision"*

		A	
B	OR	TRUE	FALSE
	TRUE	TRUE	TRUE
	FALSE	TRUE	FALSE

true when **ONE** is true

NOT A, NOT B



Inverts **true/false** value

👉 EXAMPLE:

A: Sarah has a driver's license

B: Sarah has good vision

Boolean variables that can be either TRUE or FALSE

AN EXAMPLE



BOOLEAN VARIABLES

👉 A: Age is greater or equal 20

false

👉 B: Age is less than 30

true

age = 16

LET'S USE OPERATORS!

👉 !A

false

true

👉 A AND B

false

true

false

👉 A OR B

false

true

true

👉 !A AND B

true

true

true

👉 A OR !B

false

false

false

		A	
B	AND	TRUE	FALSE
	TRUE	TRUE	FALSE
	FALSE	FALSE	FALSE

		A	
B	OR	TRUE	FALSE
	TRUE	TRUE	TRUE
	FALSE	TRUE	FALSE

THE COMPLETE JAVASCRIPT COURSE

FROM ZERO TO EXPERT!

SECTION JAVASCRIPT FUNDAMENTALS – PART 1

LECTURE JAVASCRIPT RELEASES: ES5, ES6+ AND ESNEXT

JS

A BRIEF HISTORY OF JAVASCRIPT

1995

- 👉 Brendan Eich creates the **very first version of JavaScript in just 10 days**. It was called Mocha, but already had many fundamental features of modern JavaScript!



1996

- 👉 Mocha changes to LiveScript and then to JavaScript, in order to attract Java developers. However, **JavaScript has almost nothing to do with Java** 🙅

- 👉 Microsoft launches IE, **copying JavaScript from Netscape** and calling it JScript;



1997

- 👉 With a need to standardize the language, ECMA releases ECMAScript 1 (ES1), the first **official standard for JavaScript** (ECMAScript is the standard, JavaScript the language in practice);



2009

- 👉 ES5 (ECMAScript 5) is released with lots of great new features;

2015

- 👉 ES6/ES2015 (ECMAScript 2015) was released: **the biggest update to the language ever!**

- 👉 ECMAScript changes to an **annual release cycle** in order to ship less features per update 🙏

2016 – ∞

- 👉 Release of ES2016 / ES2017 / ES2018 / ES2019 / ES2020 / ES2021 / ... / ES2089 😄

BACKWARDS COMPATIBILITY: DON'T BREAK THE WEB!

```
// ES1 Code  
function add(n) {  
  var x = 5 + add.arguments[0];  
  return x;  
}
```

1997



**BACKWARDS
COMPATIBLE**

**Modern JavaScript
Engine**

2020

DON'T BREAK THE WEB!

- 👉 Old features are **never** removed;
- 👉 Not really new versions, just **incremental updates** (releases)
- 👉 Websites keep working **forever!**

**Modern JavaScript
Engine**

2020



**NOT FORWARDS
COMPATIBLE**

```
// ES2089 Code 🤔  
c int add n <=> int 5 + n
```

2089

HOW TO USE MODERN JAVASCRIPT TODAY



During development: Simply use the latest Google Chrome!



During production: Use Babel to transpile and polyfill your code (converting back to ES5 to ensure browser compatibility for all users).

<http://kangax.github.io/compat-table>

ES5

- ➡ Fully supported in all browsers (down to IE 9 from 2011);
- ➡ Ready to be used today 👍

ES6/ES2015



ES2020

- ➡ **ES6+:** Well supported in all **modern** browsers;
- ➡ No support in **older** browsers;
- ➡ Can use **most** features in production with transpiling and polyfilling 😊

ES2021 – ∞

- ➡ **ESNext:** Future versions of the language (new feature proposals that reach Stage 4);
- ➡ Can already use **some** features in production with transpiling and polyfilling.

↖ Will add new videos

BABEL

(As of 2020)

MODERN JAVASCRIPT FROM THE BEGINNING

- 🔥 Learn modern JavaScript from the beginning!
- 👉 But, also learn how some things used to be done **before** modern JavaScript (e.g. `const` & `let` vs `var` and function constructors vs ES6 `class`).

3 reasons why we should not forget the Good Ol' JavaScript:

- 👉 You will better understand how JavaScript actually works;
- 👉 Many tutorials and code you find online today are still in ES5;
- 👉 When working on old codebases, these will be written in ES5.

JAVASCRIPT FUNDAMENTALS – PART 2

THE COMPLETE JAVASCRIPT COURSE

FROM ZERO TO EXPERT!

SECTION

JAVASCRIPT FUNDAMENTALS – PART 2

LECTURE

FUNCTIONS CALLING OTHER FUNCTIONS

JS

CALLING A FUNCTION INSIDE A FUNCTION: DATA FLOW

```
const cutPieces = function (fruit) {  
  return fruit * 4;  
};  
  
const fruitProcessor = function (apples, oranges) {  
  
  const applePieces = cutPieces(apples);  
  const orangePieces = cutPieces(oranges);  
  
  const juice = `Juice with ${applePieces} pieces of  
apple and ${orangePieces} pieces of orange.`;  
  return juice;  
};  
  
console.log(fruitProcessor(2, 3));
```

The diagram illustrates the data flow in the provided JavaScript code. Red arrows represent the flow of data from the console log to the fruitProcessor function, then to the cutPieces function, and back to the fruitProcessor function. A yellow arrow shows the flow of data from the console log to the fruitProcessor function.

- The console log `console.log(fruitProcessor(2, 3));` passes the argument `2` (highlighted in a red box) to the `fruitProcessor` function.
- The `fruitProcessor` function receives `2` as the `apples` argument and `3` (highlighted in a yellow box) as the `oranges` argument.
- Inside `fruitProcessor`, the `cutPieces` function is called with `apples` (`2`) as an argument.
- The `cutPieces` function calculates `2 * 4 = 8` and returns `8` to the `fruitProcessor` function.
- The `fruitProcessor` function also calls `cutPieces` with `oranges` (`3`) as an argument.
- The `cutPieces` function calculates `3 * 4 = 12` and returns `12` to the `fruitProcessor` function.
- The `fruitProcessor` function then constructs the `juice` string using the returned values and returns it.

THE COMPLETE JAVASCRIPT COURSE

FROM ZERO TO EXPERT!

SECTION JAVASCRIPT FUNDAMENTALS – PART 2

LECTURE REVIEWING FUNCTIONS

JS

FUNCTIONS REVIEW; 3 DIFFERENT FUNCTION TYPES

👉 Function declaration

Function that can be used before it's declared

```
function calcAge(birthYear) {  
  return 2037 - birthYear;  
}
```

👉 Function expression

Essentially a function *value* stored in a variable

```
const calcAge = function (birthYear) {  
  return 2037 - birthYear;  
};
```

👉 Arrow function

Great for a quick one-line functions. Has no `this` keyword (more later...)

```
const calcAge = birthYear => 2037 - birthYear;
```



Three different ways of writing functions, but they all work in a similar way: receive **input** data, **transform** data, and then **output** data.

FUNCTIONS REVIEW: ANATOMY OF A FUNCTION

