MENTORMATE™
DevCamp

JavaScript Fundamentals

Yulia Tenincheva

Senior Cloud Engineer, MentorMate

whoami

Yulia Tenincheva

Senior Cloud Engineer @ MentorMate

~5 years of development experience with Node.js

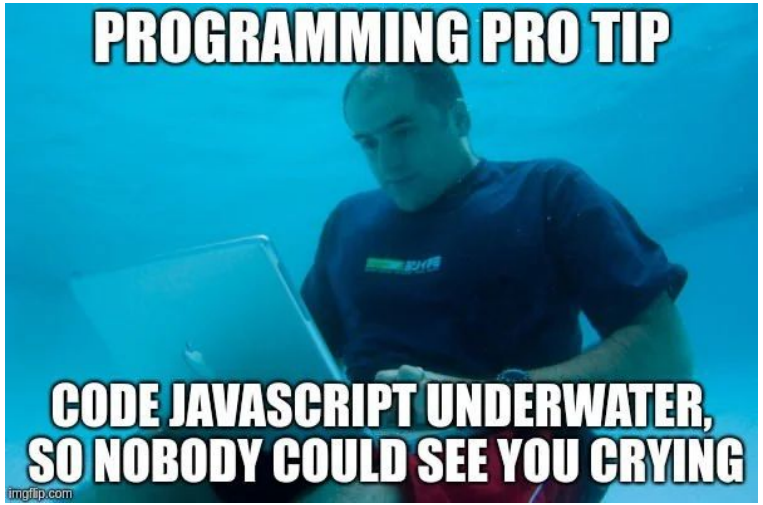
Open Source Contributor & Supporter

Agenda for today

- What you should know about JavaScript
- Dev environment setup
- Variables, scope & hoisting
- Data Types in JavaScript
- Operators in JavaScript
- JavaScript Syntax Overview

Next topics

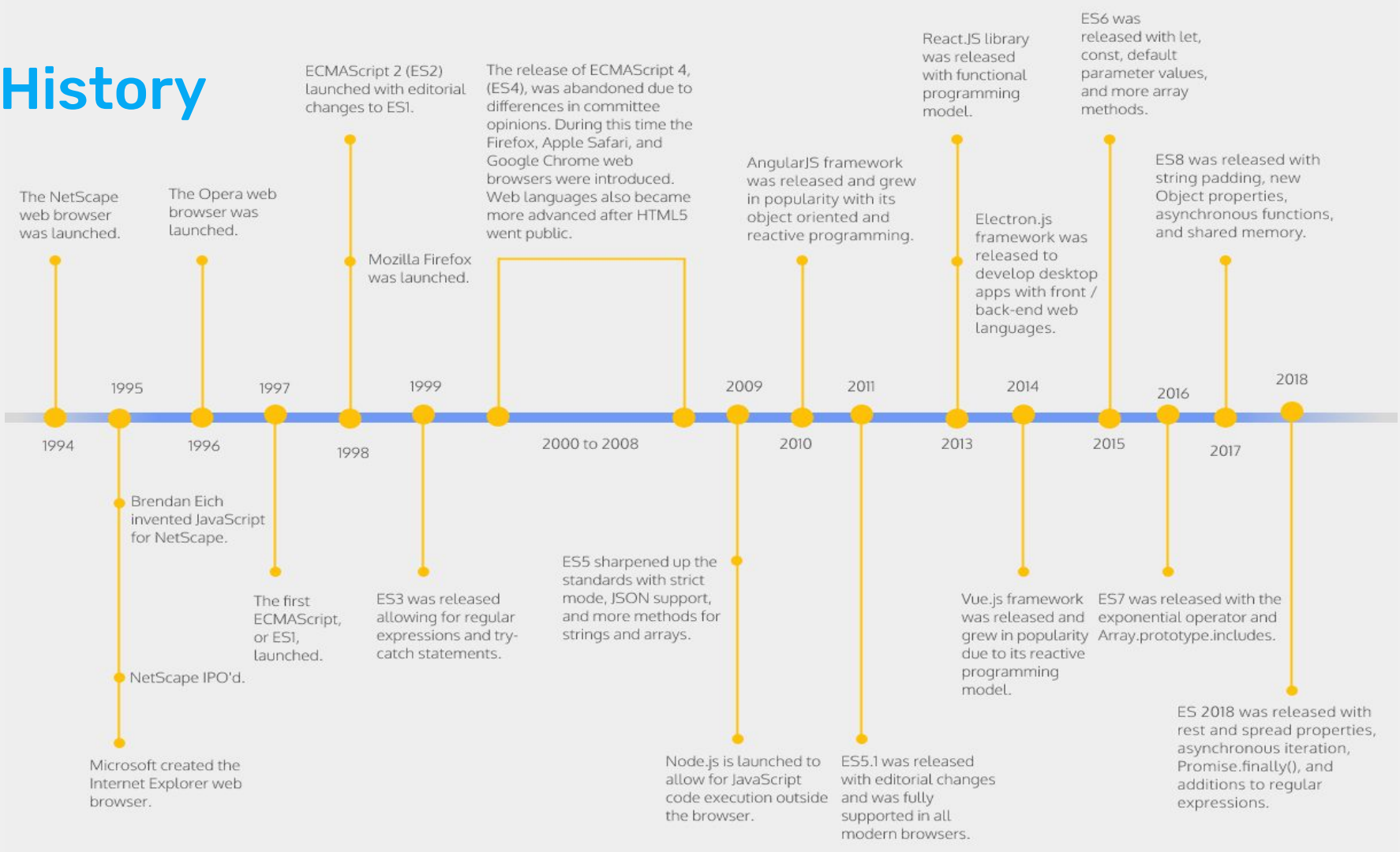
- Working with arrays and objects
- OOP vs Functional Programming
- RegExp & Text Processing
- DOM, Fetch API, JSON
- Event Loop, Asynchronous flow
- NPM, Modules
- Problem Solving



So WHAT is JavaScript?

"Modern JavaScript is a general purpose, multi-paradigm programming language that conforms to the ECMAScript specifications"

History


The NetScape web browser was launched.
The Opera web browser was launched.
ECMAScript 2 (ES2) launched with editorial changes to ES1.
Mozilla Firefox was launched.
The release of ECMAScript 4, (ES4), was abandoned due to differences in committee opinions. During this time the Firefox, Apple Safari, and Google Chrome web browsers were introduced. Web languages also became more advanced after HTML5 went public.
AngularJS framework was released and grew in popularity with its object oriented and reactive programming.
React.JS library was released with functional programming model.
Electron.js framework was released to develop desktop apps with front / back-end web languages.
ES6 was released with let, const, default parameter values, and more array methods.
ES8 was released with string padding, new Object properties, asynchronous functions, and shared memory.
1994
1995
1996
1997
1998
1999
2000 to 2008
2009
2010
2011
2013
2014
2015
2016
2017
2018
Brendan Eich invented JavaScript for NetScape.
NetScape IPO'd.
Microsoft created the Internet Explorer web browser.
The first ECMAScript, or ES1, launched.
ES3 was released allowing for regular expressions and try-catch statements.
ES5 sharpened up the standards with strict mode, JSON support, and more methods for strings and arrays.
Node.js is launched to allow for JavaScript code execution outside the browser.
ES5.1 was released with editorial changes and was fully supported in all modern browsers.
Vue.js framework was released and grew in popularity due to its reactive programming model.
ES7 was released with the exponential operator and Array.prototype.includes.
ES 2018 was released with rest and spread properties, asynchronous iteration, Promise.finally(), and additions to regular expressions.

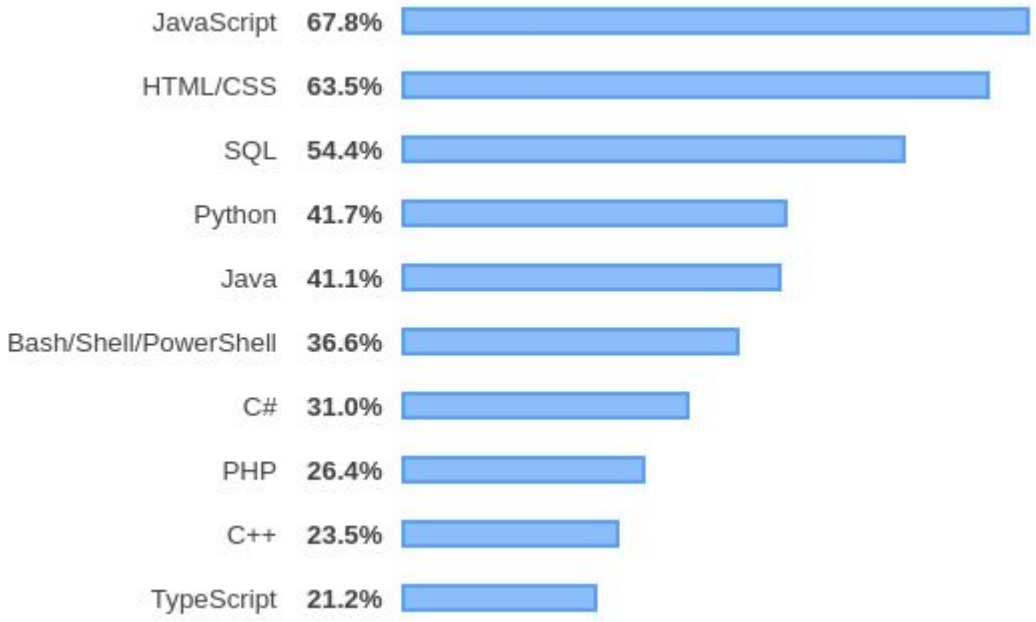
If you are into stories...

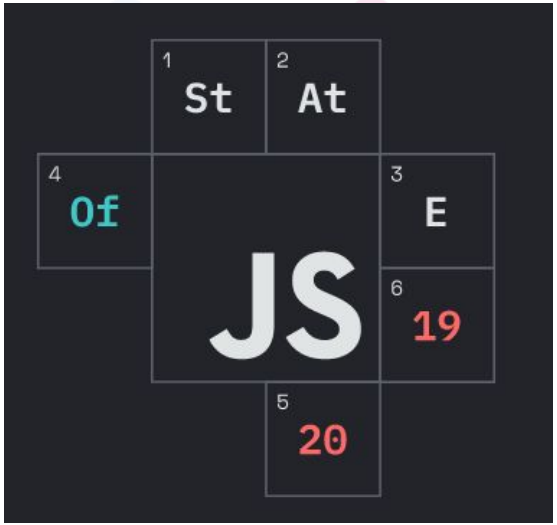
- The Weird History of JavaScript
- The Power of Paradigm by Douglas Crockford

And again....

By reading the ECMAScript specification, you learn how to **create** a scripting language. By reading the JavaScript documentation, you learn how to **use** a scripting language.

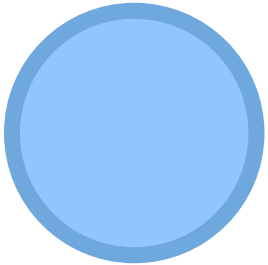
State of JavaScript today



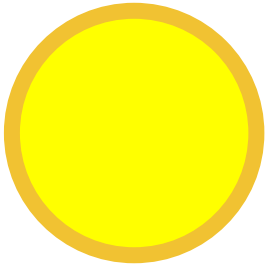


https://2019.stateofjs.com/

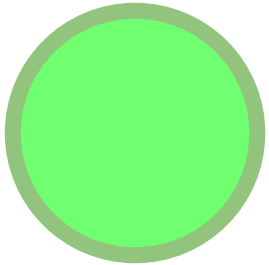
What can JavaScript do?



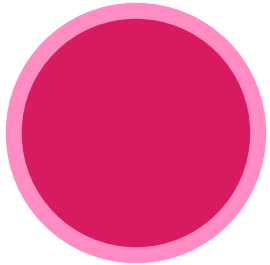
Web / Mobile Apps



Real-time Networking Apps



Command-line Tools



Games

Some creative examples: here


JavaScript
JavaScript Everywhere
SPACE RANGER
LIGHTYEAR

“Any application that can be written in JavaScript, will eventually be written in JavaScript”

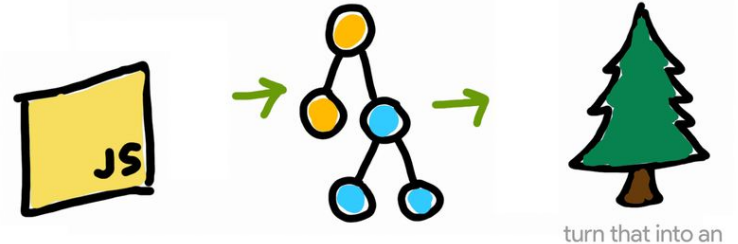
—Jeff Atwood

It’s a Teamwork!

- Some inspiration:
  - Experimental CSS Projects
  - Best HTML5 Websites
  - Ridiculously expensive Pens

But HOW ???

How V8 Works



We take your JS

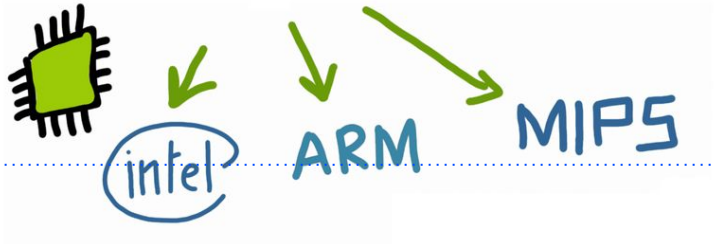
Parse it

turn that into an Abstract Syntax Tree

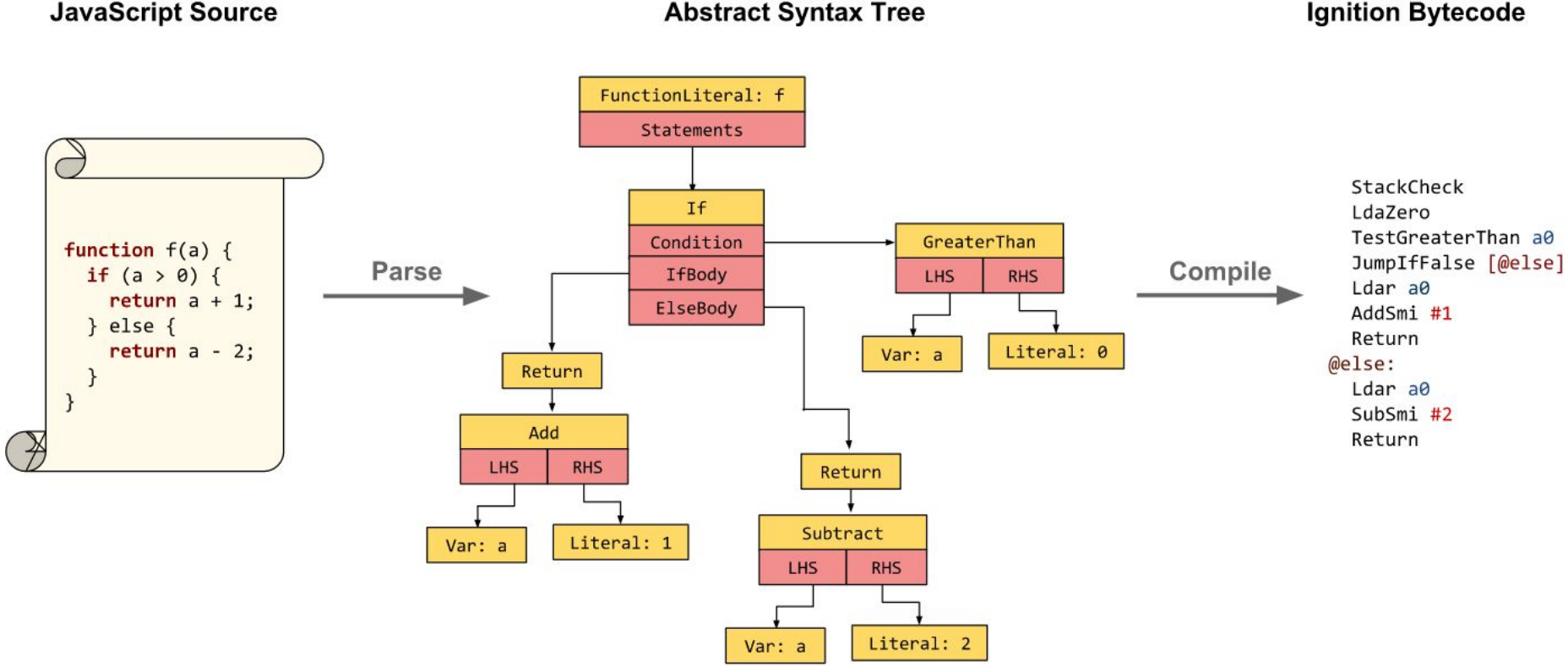
Generate bytecode

Get feedback for speculative optimisations

Optimize & Compile it



then run your optimized code!


JavaScript Source
Abstract Syntax Tree
Ignition Bytecode
function f(a) {
if (a > 0) {
return a + 1;
} else {
return a - 2;
}
}
Parse
FunctionLiteral: f
Statements
If
Condition
IfBody
ElseBody
GreaterThan
LHS
RHS
Var: a
Literal: 0
Return
Add
LHS
RHS
Var: a
Literal: 1
Return
Subtract
LHS
RHS
Var: a
Literal: 2
Compile
StackCheck
LdaZero
TestGreaterThan a0
JumpIfFalse [@else]
Ldar a0
AddSmi #1
Return
@else:
Ldar a0
SubSmi #2
Return

“Talk is cheap. Show me the code!”

—Linus Torvalds

Mandatory “Hello World” time

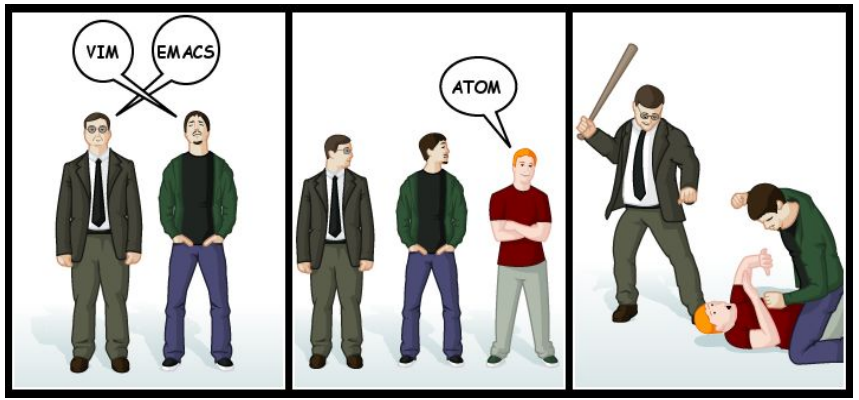
- In browser -> CTRL + Shift + i -> Console
- In HTML file, using the <script> tag
- In Node.js -> REPL
- With Node.js -> create index.js file & run node index.js in the terminal
- In any online editor

Development Environment Setup

- JavaScript Editors - choose one of Visual Studio Code, Sublime Text, Atom or smth else.

  If you feel adventurous and wanna become a *true* programmer - try Emacs or Vim *grin*

- JavaScript IDEs - WebStorm, VisualStudio
- Online editors - JSFiddle, JSBin, CodePen
- Install Node.js & NVM
- JavaScript Style Guides & Compliance



What are variables?

In JS variables are containers for storing values.

The value of a variable can be anything. Some examples include:

- a primitive type like a number, string, boolean, null or undefined
- an object
- an array
- a function

JS is weakly typed.

“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”

—James Whitcomb Riley

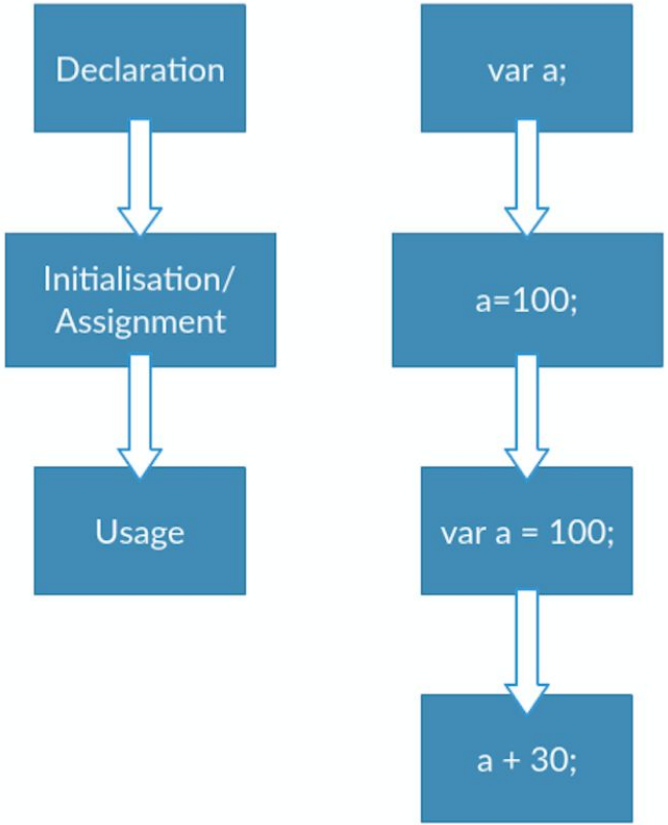
MENTORMATE™
DevCamp

Declaring a variable

Prior to ES6, variables were declared with the keyword **var**.

var is function scoped. It can be reassigned. It can also be redeclared, which isn't a desired behavior. **Do not use it in modern applications!**

```
var variable = "I'm a super cool variable";
var variable = "But I can be redeclared :(";
```



Hoisting variables

It is important to remember that in the background, JavaScript is religiously declaring, then initialising our variables!

```
function hoist() {
  a = 20;
  var b = 100;
}

hoist();

console.log(a); // 20
console.log(b); // b is not defined
```

“let” & “const” to the rescue

ES6 introduces the “let” and “const” keywords for variable declarations. They solve all the problems that “var” has.

- let - **block** scoped, **can** be reassigned, **can’t** be redeclared
- const - **block** scoped, **can’t** be reassigned, **can’t** be redeclared

Prefer to use “const”, unless you have to reassign to the same variable.

var vs **let** vs **const** **QUIZ** :))

Data Types

- Primitive Data Types
- Composite / Non-primitive Data Types
- Types comparison

Primitive Data Types

- Boolean
- Number & BigInt
- String
- Null
- Undefined
- Symbol

```
var isTrue = true; // Boolean Type

var x;             // Now x is undefined
x = 5;             // Now x is a Number
x = "Pesho";       // Now x is a String
x = null;          // Now x is Null
x = 5n;            // Now x is BigInt

var id = Symbol("id"); // Symbol Type
```

Numbers

- Numbers in JS are all doubles. In addition to always representing floating point numbers there are 3 symbolic values - +Infinity, -Infinity and NaN (not a number).

```
// Numbers
typeof 15;  // Returns: "number"
typeof 42.7;  // Returns: "number"
typeof 2.5e-4;  // Returns: "number"
typeof Infinity;  // Returns: "number"
typeof NaN;  // Returns: "number". Despite being "Not-A-Number"
```

BigInt or “the one you’re likely never going to use”

- **BigInt** is a built-in object that provides a way to represent **whole** numbers larger than 2^53 - 1, which is the largest number JavaScript can reliably represent with the Number primitive. A BigInt can be initialized by putting “n” after an integer value, or by using the BigInt constructor.

- BigInts and Numbers can’t be mixed in arithmetic operations. Math methods also don’t work on BigInts. Converting a BigInt to a Number makes it lose precision. Avoid doing that.

- In general, avoid them altogether. There are few use cases for them and unless you’re sure that you need a BigInt, don’t use it.

```
typeof 5n;    // Returns: "bigint”
```

Strings

- Strings are used to represent text. Strings in JS can be defined by using double quotes - “some string” or single quotes - ‘some string’. Both work, but It’s important to be consistent, so pick one for your project and stick with it!
- Using backticks you can define templated strings like `Hi my name is **${**name**}**` They allow expressions to be inserted anywhere in the string.
- Strings can be concatenated with the “+” operator

```
// Strings
typeof '';  // Returns: "string"
typeof 'hello';  // Returns: "string"
typeof '12';  // Returns: "string". Number within quotes is typeof string
```

Symbols

The “Symbol” function returns a unique value of type Symbol.

This means that `Symbol('foo') === Symbol('foo')` is false

The primary purpose of symbols is to be a unique identifier for object properties.

```
// Symbols
typeof Symbol();  // Returns 'symbol'
```

Primitive Data Types

```
// Booleans
typeof true;  // Returns: "boolean"
typeof false;  // Returns: "boolean"

// Undefined
typeof undefined;  // Returns: "undefined"
typeof undeclaredVariable; // Returns: "undefined"

// Null
typeof null;  // Returns: "object"
```

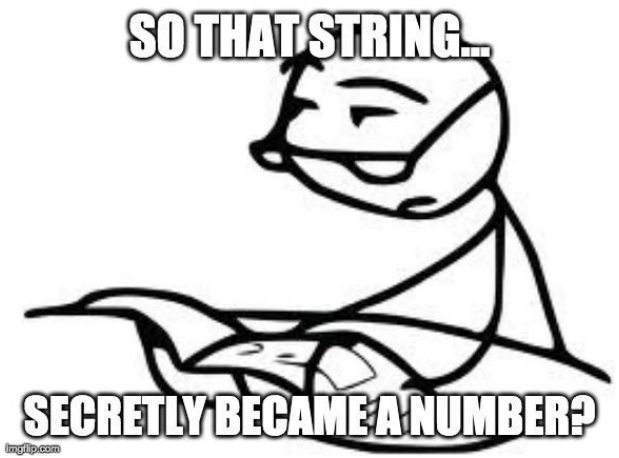
“Beware! That’s why JavaScript is tricky!”

—Your JavaScript teacher

Type coercion

Type coercion is the process of converting value from one type to another (such as string to number, object to boolean, and so on)

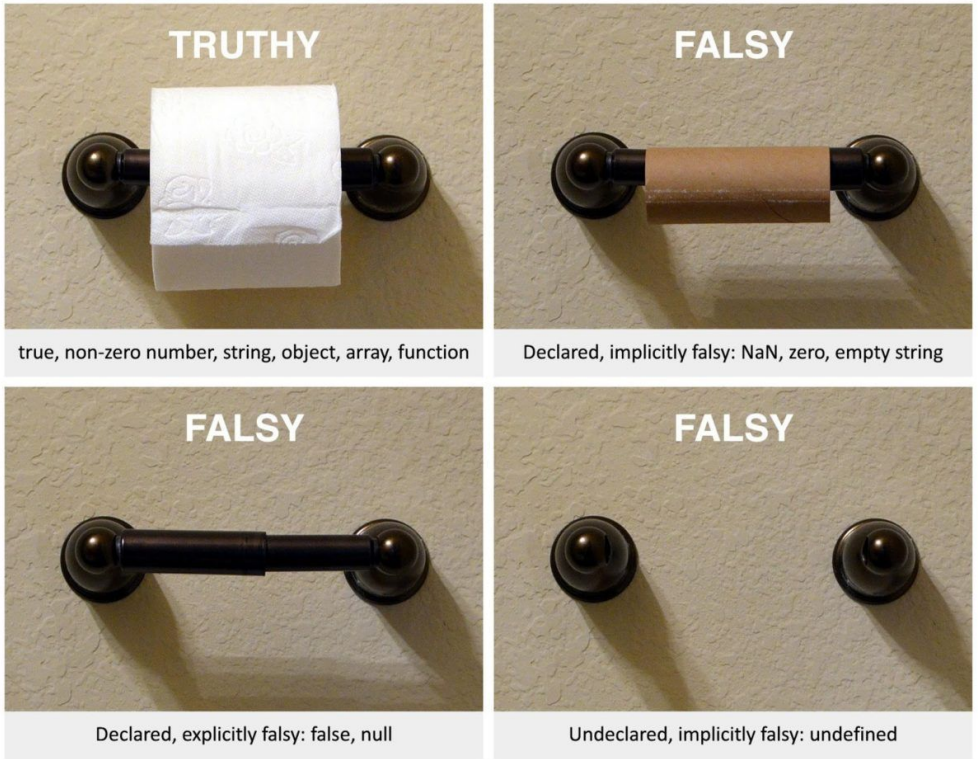
- Explicit coercion (or casting) - for example converting a string to a number by using the Number constructor `Number("123") === 123 // true`
- Implicit coercion - for example an array can get converted to a number in certain situations `1 == [1] // true`



Falsy and Truthy

JavaScript uses type conversion to coerce any value to a Boolean in contexts that require it, such as conditionals and loops.

- `false`
- `0`, `0n` (the number 0)
- `""`, `''`, ``` `` ``` (empty strings)
- `null`
- `undefined`
- `NaN`



Difference between null, undefined and ReferenceError

- **undefined** - JS assigns 'undefined' to any variable that has been declared but **not initialized**. Represents system-level, unexpected, or error-like absence of value. Also represents an array index or object property that **does not exist**.

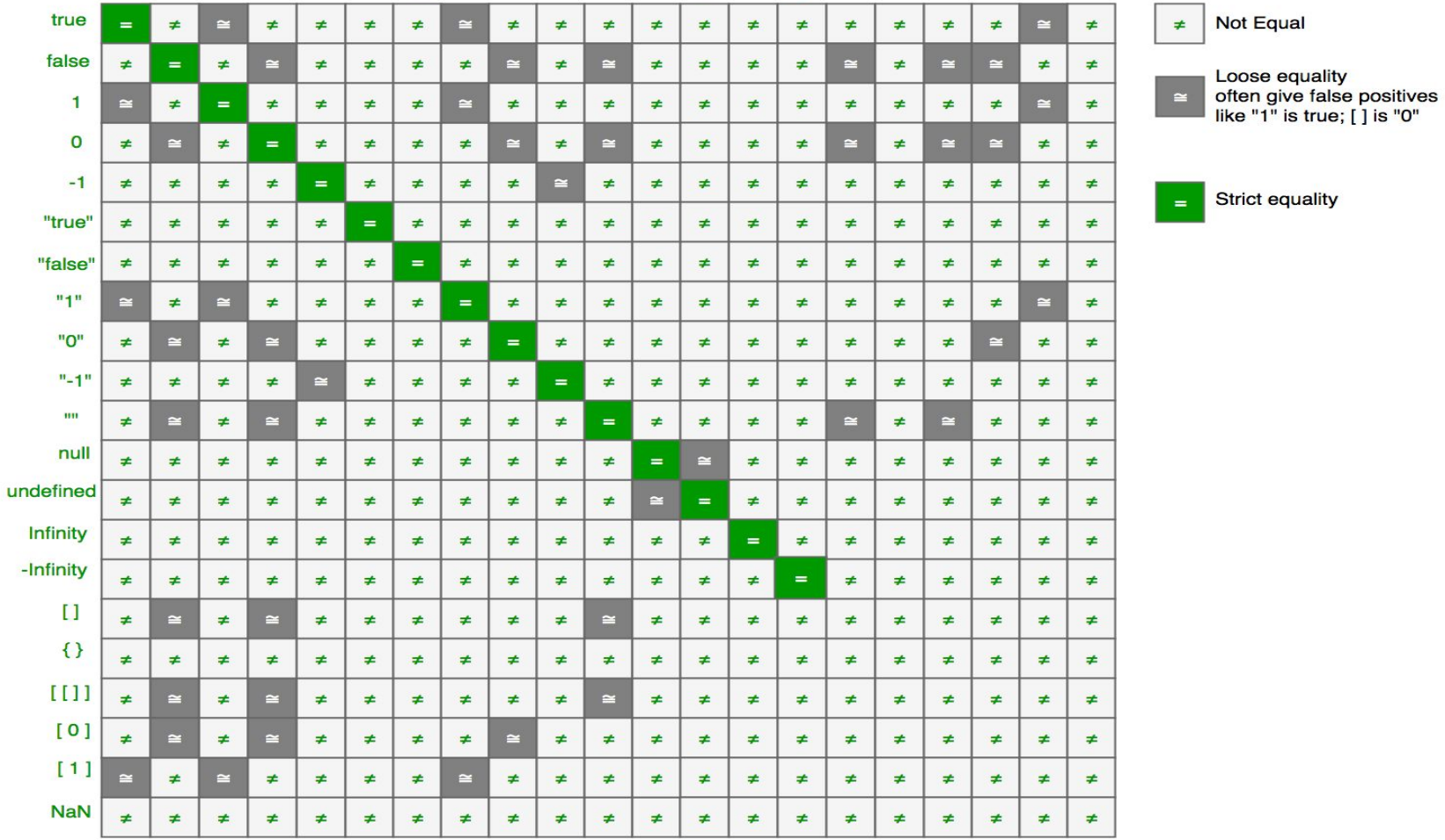
- **null** - represents the **intentional** absence of any object value. Intentional is the keyword here.

  `null == undefined // true` but `null === undefined // false`

- **ReferenceError** is thrown when trying to access a previously undeclared variable.

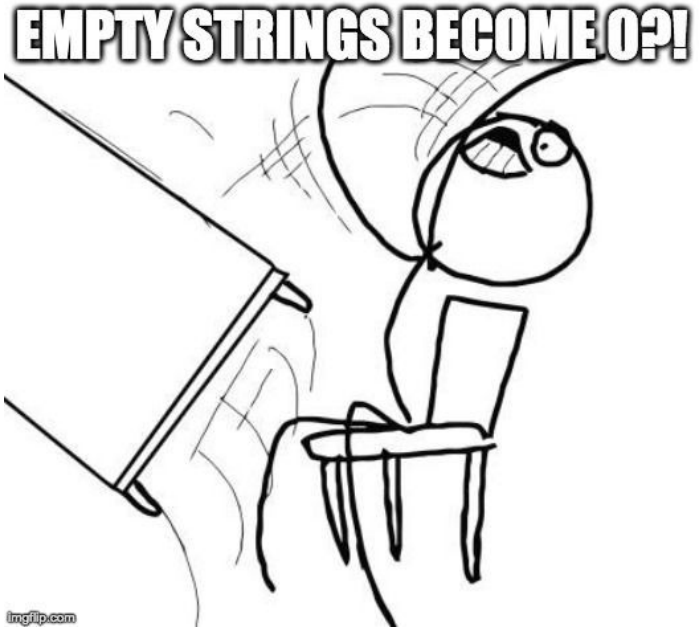
“==” vs “===”

- The “==” forces the variable to be the same and then checks their equality.
- The “===” actually requires equivalent types to give a true output.
- Same difference is valid for **!=** and **!==**


true
false
1
0
-1
"true"
"false"
"1"
"0"
"-1"
""
null
undefined
Infinity
-Infinity
[ ]
{ }
[[ ]]
[0]
[1]
NaN
Not Equal
Loose equality
often give false positives
like "1" is true; [ ] is "0"
Strict equality

Funny JS quirks

- It's a fail
- NaN is not NaN
- Adding arrays
- parseInt
- [] and null
- Magically increasing numbers
- Precision of 0.1 + 0.2
- Funny math
- Call call call



Always use “===” (triple equals / strict equality) operator to avoid unexpected behavior!

JavaScript Operators

Comparison	==, ===, !=, !==, >, <, >=, <=
Logical	&&, ||, !
Conditional (Ternary)	?, ??
Assignment	=, +=, -=, *=, /=, %=, **=, ??=
Arithmetic	+, -, *, **, /, %, ++, --
String	+, +=

Assignment operators

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Logical operators

- Logical AND (&&) - **expr1 && expr2**
  Returns expr1 if it can be converted to false; otherwise, returns expr2.
  When used with Boolean values, && returns true if both operands are true; otherwise, returns false.

- Logical OR (||) - **expr1 || expr2**
  Returns expr1 if it can be converted to true; otherwise, returns expr2.
  When used with Boolean values, || returns true if either operand is true; if both are false, returns false.

- Logical NOT (!) - **!expr**
  Returns false if its single operand that can be converted to true; otherwise, returns true.

Control Flow Statements

But first, let's introduce blocks

The most basic statement is the **block** statement, denoted by curly brackets. Its purpose is to group a bunch of statements together.

```
{
        statement_1;
        statement_2;
        ⋮
        statement_n;
}
```

Conditional statements

- **if** to specify a block of code to be executed, if a specified condition is true
- **else** to specify a block of code to be executed, if the same condition is false
- **else if** to specify a new condition to test, if the first condition is false
- **switch** to select one of many blocks of code to be executed

```
if (condition) {
  // block of code
} else if (anotherCondition) {
  // block of code
} else {
  // block of code
}
```

```
switch (operator) {
  case '+':
    // block of code
    break;
  case '-':
    // block of code
    break;
  default:
    throw new Error('Invalid Operator');
}
```

Ternary operator

```
// if (condition) {
//     variable = something;
// } else {
//     variable = somethingElse;
// }

variable = (condition) ? something : somethingElse;
```

Avoid nesting ternaries too much as it leads to **unreadable code**!

Loops

- while - loops through a block of code while a specified condition is true
- do while - loops through a block of code once, and then repeats the loop while a specified condition is true
- for - loops through a block of code a number of times
- for in - loops through the properties of an object
- for of - loops through the values of an iterable object

Each of them supports **break** and **continue**.

```
while (condition) {
  // code
}

do {
  // code
} while (condition);

for (let i = 0; i < 10; i++) {
  // code
}
```

```
for (const key in object) {
  // code
}

for (const index in array) {
  // code
}

for (const item of iterable) {
  // code
}
```

Return statement

The return statement stops the execution of a function and returns a value from that function. A return statement can return **any value**.

```
function getSomething() {
  return { foo: 'bar' };
}
```

Avoid using multiple return statements!

The throw statement

The throw statement throws (generates) an error. The technical term for this is: **throwing an exception**

- The exception can be a String, a Number, a Boolean or an Object.
- When an error is thrown, **the execution stops**
- Errors can be handled with the **try/catch** statement

```
function getOddNumbers(numbers) {
  if (!Array.isArray(numbers)) {
    throw new Error('Numbers must be an array');
  }

  return numbers.filter(x => x % 2 !== 0);
}

getOddNumbers([1, 2, 3, 4, 5]); // [1, 3, 5]
getOddNumbers('some string'); // this throws
```

Homework

Homework

1. EcmaScript 2020 specification introduces some super cool new features. List any 3 new features (that you understand) and research which JavaScript engines, browsers (and browser versions) already support them and which - not yet.

2. Explain in your words why null is an object in JavaScript, but `null instanceof Object` returns false.

3. Create sample expressions, using all possible JavaScript operators.

4. Write a JavaScript conditional statement to sort three numbers. Display an alert box to show the result. Do NOT use arrays.

Homework

5. Write a JavaScript program to display the current day and time in the following format.
Sample Output : Today is : Wednesday.
Current time is : 12 PM : 03 : 38

6. Write a JavaScript program that counts the number of workdays and holidays to the end of the year. Bonus points for including Bulgarian national holidays.

7. Write a JavaScript program to find when your birthday is on Friday between 2020 and 2050.

Homework

8. FizzBuzz. Write a program that prints the numbers from 1 to 100. But for multiples of three print “Fizz” instead of the number and for the multiples of five print “Buzz”. For numbers which are multiples of both three and five print “FizzBuzz”. Try to do that with the least amount of code possible.

9. Write a JavaScript programs to convert the following metrics:
- Temperatures to and from Celsius, Fahrenheit
- mph to and from km/h
- Gallons to and from litres

10. Write a JavaScript program to check from two given integers, whether one is positive and another one is negative.

Homework

11. Write a JavaScript function to calculate the tax (20%) of a given price.

12. Write a JavaScript function to round a number to a given decimal places. Do NOT use string methods or arrays.
Test Data :
console.log(precise_round(12.375,2));
console.log(precise_round(12.37499,2));
console.log(precise_round(-10.3079499, 3));
Output :
"12.38"
"12.37"
"-10.308"

Practice, Practice, Practice!