BO·TT·EGA
IT minds

Training program:

JavaScript deeper - for those who want to understand JS

Info:

Name:	**JavaScript deeper - for those who want to understand JS**
Code:	**JS-deeper**
Category:	JavaScript
Target audience:	developers
Duration:	3-4 days
Format:	20% lecture / 80% workshop

BO·TT·EGA
IT minds

Training program

1. Types - why most of the type coercion explanations are too simplified

1.1. Implementing conversion algorithms for ToBoolean and ToPrimitive

1.2. Building a general conversion model

1.3. Learning when should we use coercion

1.4. Building the internal algorithm of ==

2. Object-oriented programming - why classical OOP is unnecessarily complex in JS

2.1. Building the complex mental models for constructor/class-based programs

2.2. Building the complex mental models for programs based on objects linking

2.3. Distinguishing .prototype, [[Prototype]] and __proto__

2.4. Knowing new and instanceof from the inside

2.5. Discovering nuances of Object.assign copying

2.6. Identifying leaky class abstractions

3. Functions and its scopes - embracing its powers

3.1. Building more detailed hoisting understanding by manipulating the call context

3.2. Building more detailed scope understanding based on call stack

3.3. Nuances of block scope - differences between const, let and var

3.4. Functions piece by piece - arguments vs parameters, default values, variable number of arguments

3.5. Comparing 4 dynamic this binding methods

3.6. Testing and comparing the performance and memory footprint of closures and prototypes

3.7. Looking for the cases when arrow functions should not be the direct replacements of ES5 functions

4. Functional programming I - use it even sparingly and get the simpler and more reusable code

4.1. Discussing what the functional programming really is

BO·TT·EGA
IT minds

4.2. Implementing built-in higher-order functions (map/filter/reduce/find)

4.3. Implementing higher-order functions missing in JS (flatMap, zip, takeWhile etc.)

4.4. Analyzing the ways to keep the objects immutable (seal, freeze, deepFreeze etc.)

4.5. Optimizing function calls using memoization

5. Asynchronous JS - consciously choosing the proper way to handle asynchronicity

5.1. Analyzing the event loop-based concurrency model

5.2. Finding problems with callbacks other than nesting

5.3. Building utils for Promises - finally, timeout, first, retry, collapse

5.4. Using async/await to make the asynchronous code look synchronously

6. Modularity - maximizing our options in module management

6.1. Analyzing Revealing Module and Dependency Injection patterns

6.2. Implementing own EventEmitter to better understand the Observer pattern

6.3. Adding choreography using EventEmitter

6.4. Building the understanding of require, exports and module.exports of CommonJS

6.5. Comparing ES6 modules with CommonJS

7. Functional programming II - food for thought for FP fans

7.1. Differentiating between partial application and currying

7.2. Replacing object-oriented Dependency Injection based on constructors with function-oriented Dependency Injection based on currying

7.3. Meeting pros and cons of point-free programming

7.4. Building an abstraction to compose functions into blocks: compose and pipe

7.5. Replacing null and undefined with composable Maybe

7.6. Replacing try/catch with composable Either