Netcracker

Front-end development course
JavaScript

Lecture #1

Let's get acquainted

Nikolay Kluyev, Software Engineer from Netcracker corp.

Contacts:

nikolay.klyuev@netcracker.com

https://t.me/hartex

kluyev.nikolay

Course introduction

Introduction to JavaScript

- Not related to Java (almost).
- JS - one of the most popular languages and most popular language on GitHub.
- Interpreted, multiparadigmatic, dynamically typed, with automatic memory management and garbage collection.
- Doesn't have standard library like Java or C++.
- In JavaScript - functions are first class citizens: language allows to use higher order functions, anonymous and inner functions.
- JS is based on ECMAScript specification (first standard came out in 1997).
- There are many languages that can be compiled to JS (TypeScript, Dart, CoffeeScript, Elm (even Java can) and etc), but browser can only understand JavaScript.

© NetCracker 2018

JavaScript most important releases and specs

1	June 1997	First, initial edition
3	December 1999	RegExp, better string handling, new control statements, try/catch, formatting for numeric output
5	December 2009	"strict mode", getters and setters, support for JSON
6	June 2015	Also known as “ES6”, “ES2015”, “ECMAScript Harmony”. Features added: classes and modules, iterators and for/of loops, generators and generator expressions, arrow functions, binary data, typed arrays, collections (maps, sets and weak maps), promises, number and math enhancements, reflection, and proxies (metaprogramming for virtual objects and wrappers).

Latest ECMAScript specification can be found here
ES6 browser support https://kangax.github.io/compat-table/es6/

IDE and text editors

You definitely need one.

IDE’s:

- JetBrains products (IntelliJ IDEA, WebStorm and others).
- Eclipse with plugins.
- Visual Studio
- Komodo IDE

Text Editors:

- Visual Studio Code
- Atom
- Sublime Text
- Vim, Emacs

Basic concepts

© NetCracker 2018

Console & Debugging

- console.dir()
- console.dirxml()
- console.log()
- console.error()
- console.info()

(logging debug information in production is **a bad** practice, don’t do it)

Useful links:

- https://mozilladevelopers.github.io/playground/debugger
- https://habrahabr.ru/post/114483/
- https://developers.google.com/web/tools/chrome-devtools/javascript/

Tag <script> and script loading

Tag <script>:

- must contain an executable code
- can be placed anywhere inside page
- browser stops page parsing when it runs into <script> tag and starts loading script (if ‘src’ attribute is provided) or executing code inside tag
- take a look at the order of execution in script-basics.js file

© NetCracker 2018

External scripts

- If “src” attribute is present <script> tag content is ignored
- If we are dealing with external script browser will not show content after <script> tag until it loads that external script, evaluates and executes it.

What can we do to resolve that problem?

- Put all <script> tags at the bottom of the page
- Use “defer” attribute
- Use “async” attribute

“defer” and “async” both allow loading scripts asynchronously, but with two differences: “defer” will preserve scripts order and guarantee that script will only be executed after the document has been parsed, but before DOMContentLoaded.

More on script attributes:
https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script

© NetCracker 2018

Script loading events

- **script.onload** - triggers when browser finished loading and executing external script.
- **script.onerror** - triggers on any error during script loading (but not executing)

That events works also for **<img>** and **<link>**.

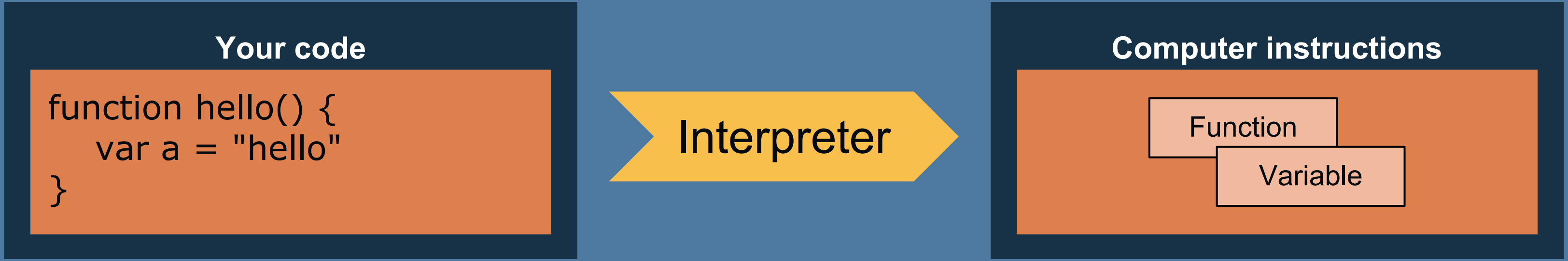
DOM loaded events:

- **DOMContentLoaded** - means that all DOM nodes are ready, you can query or add an event handler on them, but pictures or styles may not be fully loaded. Browser waits all scripts (except “async” and “defer”) to load an execute then fires **DOMContentLoaded**
- **window.onload** - fires when all content is fully loaded

Script execution concepts

Syntax parser - a program that reads your code and determines what it does and if its grammar is valid.

Parser reads your code character by character



```
function hello() {
   var a = "hello"
}
```

Script execution concepts

Esprima - one of the parsers that is written in JavaScript for JavaScript.

Such parsers can also be used in syntax highlighters, static analyses, translators, compilers, obfuscators.

Dead code elimination is super easy to do with that kind of tools.

Esprima uses the common recursive descent approach and it's output is compatible with Mozilla SpiderMonkey Parser API (there is also a separate spec for this format https://github.com/estree/estree)

You can play with Esprima on node.js, it's public API contains 3 methods:

- esprima.tokenize(input, config)
- esprima.parseScript(input, config, delegate)
- esprima.parseModule(input, config, delegate)

More information can be found in official doc

Other parsers: acorn, Narcissus from Mozilla

Script execution concepts

Parsers does two kind of operations: producing Abstracting Syntax Tree (parsing or syntactic analysis) and Tokenization (lexical analysis)

- syntactic analysis - takes a string representing a program and produces a syntax tree, an ordered tree that describes the syntactic structure of the program.
- lexical analysis - takes a string as an input and produces an array of tokens, a list of object representing categorized input characters.

XML based language is itself an AST and DOM manipulation is AST transformation.

Transformation example.

The real world example of Lexer and Parser in JavaScript engine can be found in JavaScriptCore (part of WebKit engine) repository https://trac.webkit.org/browser/webkit/trunk/Source/JavaScriptCore/parser

Links & resources

Examples source code: https://github.com/hartex/nc-js-course

Syntax parser: live demo and source code

Probably, best book series about JS: https://github.com/getify/You-Dont-Know-JS

Mozilla Developers Network: https://developer.mozilla.org/ru/

Examples source code: https://github.com/hartex/nc-js-course or http://bit.ly/2DnmVkv

Q&A

Thank You

Netcracker