JavaScript

Podstawy programowania

console.log('Kilka słów o mnie');

Michał Jabłoński

Front – End deweloper od 2007 r.

2007	(X)HTML + CSS + JavaScript
2009	Flash + HTML + AS 2.0 + PHP
	…
2011	Flex + AS 3.0 + Java EE + GWT + JavaScript
	…
2017	Angular + React + Vue + JS ES6
	+ TypeScript + Node.js

https://github.com/michaljabi

Szczegóły techniczne co do zadań:

- Generalnie do przykładów ćwiczeń wykorzystywać będziemy:
  - Visual Studio Code + Node.js + JavaScript REPL albo Quokka.js
  - W tym układzie będziemy od razu widzieć gdzie są nasze „console.log'i"
- Zadania zostały przygotowane tak, że można rozwiązywać również w środowiskach:
  - StackBlitz : https://stackblitz.com
  - JSBin.com : http://jsbin.com
  - Wystarczy je do nich przekopiować

JavaScript
dlaczego właśnie JavaScript ??!

- W skrócie: „ponieważ to język, który rozumie przeglądarka”
  - HTML + CSS + JS

- Wzrost znaczenia stron typu SPA
  oraz frameworków Front-End’owych opartych o JavaScript

- Rozwiązanie problemu statycznego dołączania skryptów do storny HTML,
  tzw. **modularyzacja JavaScript**

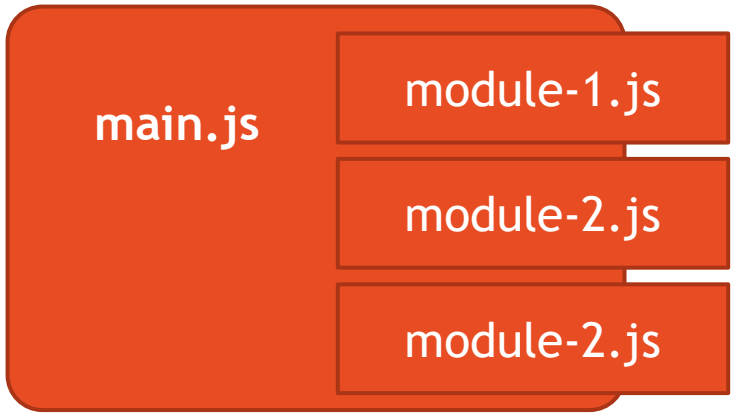
JavaScript – trochę się pozmieniało

- JS powstaje z myślą o dynamicznych modyfikacjach DOM
  - Ilość kodu potrzebna do napisania i sposób jego utrzymywania w stosunku do uzyskiwanych „dynamicznych” interakcji na stronie – skutecznie zniechęca do pisania czegoś więcej jak tylko wariatory do formularzy
- Później: pojawia się AJAX – można dynamicznie przeładowywać cześć strony
- Później: nastaje era jQuery: usystematyzowane i uproszczone API do zarządzania DOM i AJAX
- W 2009 powstaje Node.js (Server Side JavaScript)
- Po 2015: JavaScript dostaje dużo lukru składniowego i od tej pory jest odświeżany co roku.
  - Dzięki połączeniu środowiska developerskiego w Node.js i składni z 2015 dostajemy język w którym można pisać aplikacje w sposób podobny do np. Java czy C#

Co się stało z JavaScript?

Kontekst historyczny

- W 2009 pojawia się „Node.js” – który zmienia sposób pisania kodu



- Po 2015 pojawiają się „cukiereczki” tzw. Sintactic sugar / Lukier składniowy

class | Set() | Map() | …spread | `interpolacja ${x}`

Bardziej jak…
C#
JAVA

Nowe możliwości: ES6, ES7, ES8, ES-next…

- Modułowość
- Leksykalne deklarowanie zmiennych: let, const
- Funkcje Arrow
- Domyślne wartości dla argumentów funkcji
- Składniowy lukier dla klas
- Destrukturyzacja obiektów i tablic
- Operatory: ...spread i ...rest
- Interpolacja tekstu
- Generatory
- Operatory: async / await
- I inne...

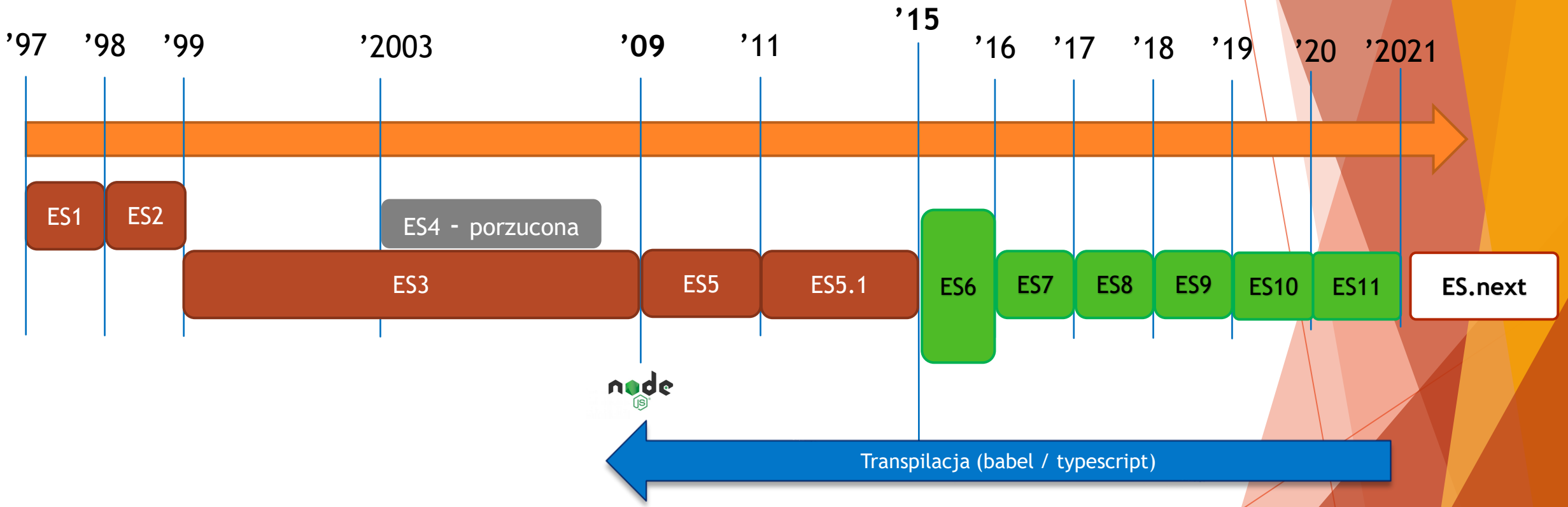
Istotne koncepcje języka nie ulegają zmianie

- Prototypowalność
- Funkcje jako First Class Citizens
- Zasięg zmiennych
- Kontekst wywołania
- Obiektowa natura JavaScriptu

Co należy wiedzieć?

- Od 2015 do JavaScript dochodzą nowości praktycznie co roku
- Specyfikacja JS opisana jest pod nazwą ECMAScript – dlatego właśnie różne wersje JS mają skrót np. ES5, czasem możesz spotkać się z zapisem używającym roku wydania np. EcmaScript2015 lub ES2015
- Do 2015 o JS mówimy jako ES5 – to wersja, którą będzie obsługiwało najwięcej przeglądarek (kompatybilność wsteczna).
- W roku 2015 wychodzi ES6.
- W 2016 dochodzą nowości i mamy ES7
- Potem w 2017 -> ES8… itd…
- ….
- W tym roku 2020 – będziemy mieć wersje ES11
- Roboczo na „kolejną wersję JavaScript” mówi się po prostu ES-next

Wersje JavaScript



- **ES** = *EcmaScript* - oznaczenie wersji *JavaScript*
- Od 2015 roku, rok rocznie mamy syntaktyczne nowości

Co to jest Transpilacja?

- OK, korzystamy z super rzeczy po 2015, język jest „odświeżony” są nowości.
- A co z kompatybilnością wstecz ?
- A co jeśli ja lubię korzystać z IE9 ?

- Zgadza się, potrzebujemy dodatkowego kroku: Transpilacji.
  Zachowania poziomu abstrakcji języka JS ale w składni np. z przed 2015r.

Rozwój JavaScriptu

- Dobrze opisane: https://en.wikipedia.org/wiki/ECMAScript
- Nowa składnia: **ES6 … ESNext**
  - transpilery pozwalające używać tej składni już dzisiaj

- Syntactic sugar do wielu rzeczy
- Nowe metody dla natywnego kodu
- Transpilery czyli np: Babel, TypeScript

Transpilacja kodu

- Inne słowo na „kompilacja” z tą różnicą, że po procesie transpilacji mamy język na podobnym poziomie abstrakcji (np. dalej edytowalny kod)

- Dlaczego to potrzene ?
  - JavaScript jest językiem skryptowym który jest „interpretowany” przez przeglądarkę
  - Po 2015 wprowadzono nowy „syntaks” JavaScriptu – część przeglądarek obsługuje już praktycznie wszystkie nowości z ES6, 7, 8 !
  - Niestety nie wiemy jaką przeglądarką dysponuje nasz EndUser i nie zawsze możemy na nim „wymusić” używania konkrenej

  - Efekt: potrzebna transpilacja do ES5!

Vanilla JavaScript

- Tak naprawę to zwykły JavaScript – bez dodatków
- Nazwa wymyślona przez JS community, miała imitować „kolejną superblibiotekę potrzebą do JavaScript”
- W rzeczywistości całość „dokumentacji” dla biblioteki przedstawia natywne metody JavaScript’u
- Dlaczego warto o tym wiedzieć
  - ponieważ zadając pytanie na **stackowerlow** często pojawia się ten zwrot i oznacza nic innego jak potrzebę napisania czegoś natywnie, bez użycia dodatkowych bibliotek typu: lodash, jquery, underscore itd..

Przygotowanie do działania

Środowisko / Ekosystem

back-end

manager pakietów / bibliotek

interpreter

front-end

Narzędzia ekosystemu / Utils

Statyczny analizator kodu
Biblioteka + CLI (Command Line Interface)

Pozwala pisać „lepszy kodzik” – bo wymusza na nas reguły pisowni oraz dobre praktyki
Integracja z IDE pozwala je wychwycić już w momencie pisania.

Transpiler kodu
Biblioteka + CLI (Command Line Interface)

Pozwala na „podróż w czasie” – z nowego kodu JS
Robimy taki który jest kompatybilny
ze starszym przeglądarkami

Biblioteki / Libs

Biblioteka do manipulacji DOM

Realizacja pomysłu jednakowego interfejsu do zarządzania DOM i odczytywania BOM. Szczególne znaczenie biblioteka odcisnęła w momencie gdy nastąpił BOOM w świecie przeglądarek WWW i każda z nich oferowała delikatne różnice w sposobie dostępu do zasobów etc. Można ją traktować jako swoistego rodzaju „Adapter” Upraszcza wykorzystanie selektorów, tworzenie Node’ów w DOM oraz zapytania AJAX

Lo

Lodash – Biblioteka

Tzw. Util library.
Niezależnie od środowiska - dostarcza funkcji i metod, nienapisanych w zwykłym JavaScript a pomocnych w codziennych aplikacjach.

- Biblioteki czasami są zależne od ekosystemu / środowiska.
- Głównie z powodu ich zastosowania.
- Przykładowo jQuery – manipuluje **DOM** – co oznacza że potrzebujemy uruchomić kod z jQuery w środku przeglądarki. Nie zadziała on bezpośrednio w Node.js

Język / składnia / ECMAScript

- Dopiero od 2015 roku widać faktyczny rozwój JavaScript.
- Składnia nie zmienia się całkowicie z roku na rok – jednak rozszerza swoją funkcjonalność.
- Jest to najbardziej sensowny rozwój JavaScript który, w połączeniu z Transpilacją i Polyfill'ingiem - zachowuje wsteczną kompatybilność z istniejącymi aplikacjami napisanymi w JavaScript
- Za rozwój standardu ECMAScript odpowiada organizacja TC39

Jak buduje się wielkie aplikacje *JavaScript* w 2020 ?

- Niezależnie od wielkości jak i tego czy to aplikacja front czy back –endowa, Node.js jest numerem 1 dla nowych projektów.

- TAK – TAK! W Node, możemy robić front end za pomocą tzw. „Bundlerów” – dzięki którym, połączone ze sobą pliki (za pomocą importów/exportów) zbierane są razem i wystawione jako jedna paczka. Bundler to specjalna biblioteka – działająca tylko podczas dewelopmentu i przygotowania produkcji.
- Dodatkowo podczas dewelopmentu inne paczki dostarczone do bundlerów, są w stanie wystawić nam lokalny serwer deweloperski – w ten sposób mamy dostęp do DOM.