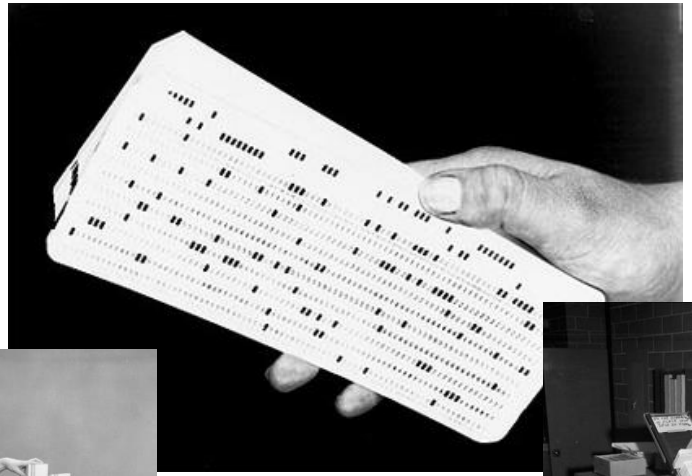
Podstawy Programowania

Na przykładzie JavaScript

Tomasz Nastały

Pierwsze języki - Fortran



1954 – 1957

FORTRAN SPECIALIST GROUP
BCS
THE BRITISH COMPUTER SOCIETY

Example code - Fortran 90 & 95

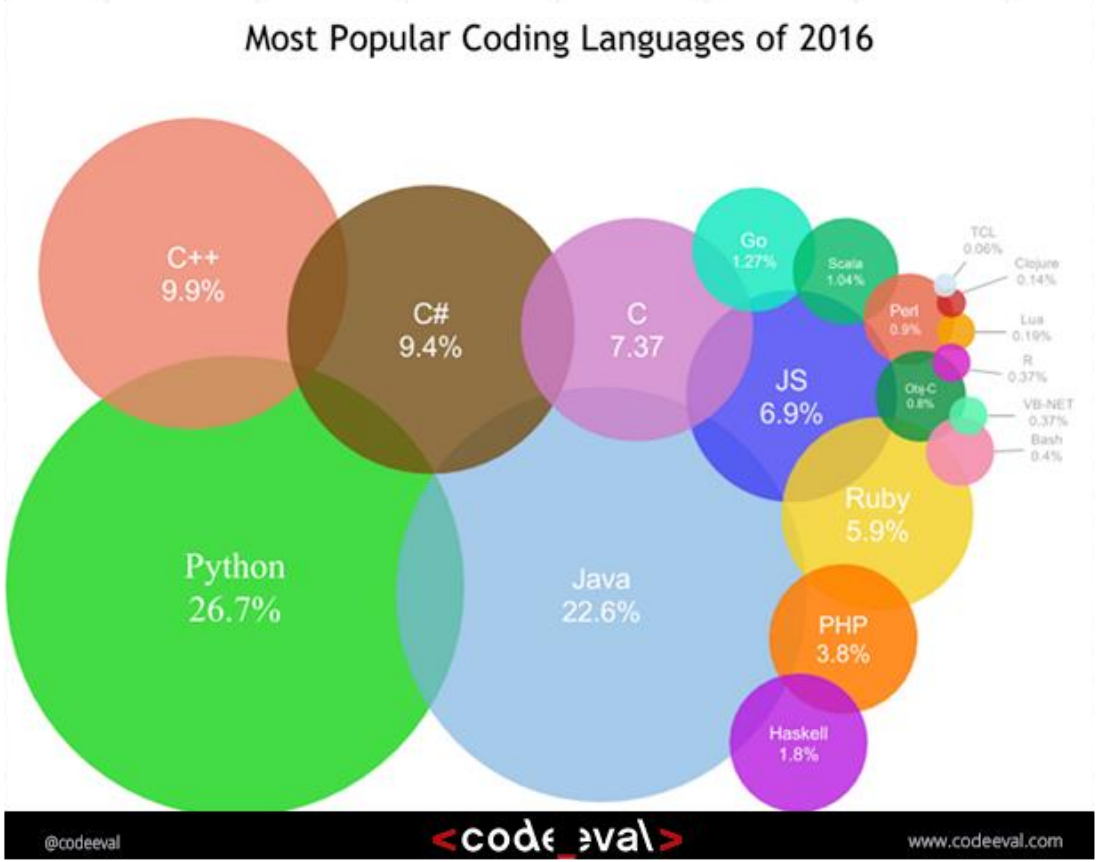
```
 PROGRAM TPK
! The TPK Algorithm
! Fortran 90 style
  IMPLICIT NONE
  INTEGER                   :: I
  REAL                      :: Y
  REAL, DIMENSION(0:10)     :: A
  READ (*,*) A
  DO I = 10, 0, -1          ! Backwards
    Y = FUN(A(I))
    IF ( Y < 400.0 ) THEN
      WRITE(*,*) I, Y
    ELSE
      WRITE(*,*) I, ' Too large'
    END IF
  END DO
  CONTAINS                   ! Local function
    FUNCTION FUN(T)
      REAL  :: FUN
      REAL, INTENT(IN) :: T
      FUN = SQRT(ABS(T)) + 5.0*T**3
    END FUNCTION FUN
  END PROGRAM TPK
```

Nie tylko JavaScript

- Asemblery (oprogramowanie procesorów)
- C (1969r., Linux, Windows, sterowniki)
- C++ (1983r., MicrosoftOffice, PDF Reader, Firefox, Autocad, gry np. Wiedzmiń 3)
- Objective-C (1984r., aplikacje na iOs i Maca)
- Ruby (1993r., Basecamp)
- Python (1994r., Youtube)
- PHP (1995r., Wordpress, Facebook, fora internetowe)
- Java (1995r., Webstorm, Android, popularny na backendzie)
- C# (2000r., Stackoverflow, popularny na backendzie)

W dużych projektach, często korzysta się z paru języków (różne części aplikacji, pisane w różnych językach).

Języki



http://blog.codeeval.com/codeevalblog/2016/2/2/most-popular-coding-languages-of-2016

Języki - podobieństwa

JAVA / C#

```
public static int sum(int a, int b) {
    return a + b;
}
```

JavaScript

```
function sum(a, b) {
    return a + b;
}
```

TypeScript (nadzbiór języka JavaScript)

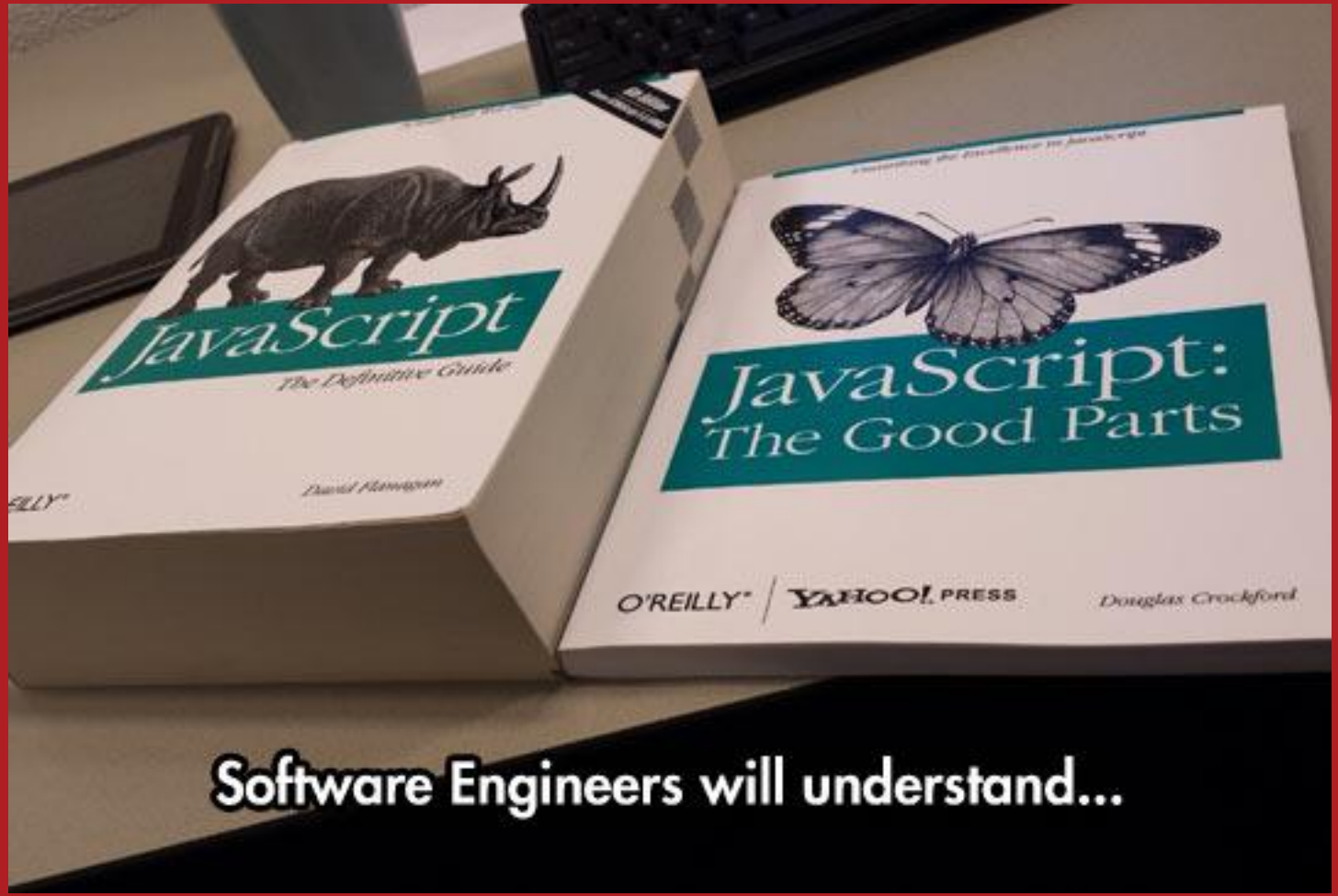
```
public static sum(a: number, b: number) : number {
    return a + b;
}
```

Objective-C

```
(NSInteger)sum:(NSInteger)ab:(NSInteger)b {
    return [a integerValue] + [b integerValue];
}
```

HelloWorld w stu językach:
https://github.com/leachim6/hello-world

Podstawy JavaScript
21.04.2018


JavaScript
The Definitive Guide
David Flanagan
JavaScript:
The Good Parts
O'REILLY
YAHOO! PRESS
Douglas Crockford
Software Engineers will understand...

1.
JavaScript - historia

JavaScript - historia

- Stworzony w 1995 roku przez Brendana Eicha
- Napisany w 10dni
- Rozwijany do dziś
- Obecnie możemy w nim również pisać backend za pomocą (NodeJS)
- Używany głównie do obsługi zachowania stron internetowych i aplikacji webowych
- Każda przeglądarka ma swój silnik potrafiący obsłużyć kod JS

EcmaScript – STANDARD
JavaScript - Implementacja

Year	Name	Description
1997	ECMAScript 1	First Edition.
1998	ECMAScript 2	Editorial changes only.
1999	ECMAScript 3	Added Regular Expressions. Added try/catch.
	ECMAScript 4	Was never released.
2009	ECMAScript 5	Added "strict mode". Added JSON support.
2011	ECMAScript 5.1	Editorial changes.
2015	ECMAScript 6	Added classes and modules.
2016	ECMAScript 7	Added exponential operator (**). Added Array.prototype.includes.

2.
JavaScript - zastosowanie

JavaScript - zastosowanie

- Interaktywne strony www
- Backend, np. tworzenie REST API (framework nodeJS)
- Skalowane aplikacje webowe (framework Angular / React / Vue)
- Aplikacje desktopowe (framework Electron, znane apki napisane w JS to np. Spotify, Slack)
- Gry (HTML5 + JS / jQuery)

3.
JavaScript - cechy

JavaScript - definicja

- Język programowania obsługiwany głównie przez przeglądarki
- Wspiera programowanie funkcyjne jak i obiektowe (i spaghetti :)
- Często używany w kombinacji z CSS i HTML
- Wysoko-poziomowy
- Dynamiczny
- Nietypowany
- Interpretowany podczas run-time’u (nie wymaga kompilacji)

4.
JavaScript – co mamy do dyspozycji

JavaScript - API

- JavaScript zawiera ogromny zbiór gotowych metod i obiektów

https://developer.mozilla.org/en-US/docs/Web/JavaScript

Najbardziej popularne:

- Metody do operacji na danych (stringach, liczbach, obiektach, tablicach)
- Metody do operacji na drzewie DOM
- Metody do obsługi eventów przeglądarki (click, mouseover, scroll)
- XHR (obsługa zapytań HTTP)

Podłączenie skryptu

<script src=”filepath/file-with-script.js”></script>

Pliki podłączamy przed zamknięciem tagu body w pliku HMTL

Konsola

Narzędzie deweloperskie -> konsola

```
console.log('hello’);
console.warn('warning')
console.table([1,2,3,4]);
console.table([[1,2,3],[4,5,6],[7,8,9]])
```

Ciekawostka:
Text w konsoli można kolorować:

```
console.log('%c Oh my god!', 'background: #222; color: #bada55','more text');
```

https://stackoverflow.com/questions/7505623/colors-in-javascript-console

5.
JavaScript – zmienne

ZMIENNA

Zmienna – kontener do przetrzymywania danych, np. wartości liczbowych lub tekstowych

Deklarowanie zmiennych

```
var firstName = "John";
var lastName = 'Doe';
var age = 23;
var isAvailable = true;

var firstName = "John",
    lastName = 'Doe',
    age = 23,
    isAvailable = true;
```

Zmienne – typy prymitywne (primitive data)

```
var firstName = "John"; // string
var age = 23; // number
var isAvailable = true; // boolean
var salary = null; // object ?!!
var job; // undefined
```

Zmienne – typy złożone (complex data)

```
var address = {
  street: 'Jana Pawła II',
  city: 'Gdańsk'
}; // object

var skills = ['HTML5', 'JavaScript', 'CSS']; // object

var getFullName = function () {
  return firstName + surName;
}; // function
```

Zmienne – brak słówka var przy deklaracji

```
name = "John"; // właściwość obiektu window, pseudo
globalna, możne zostać usunięta

var name = "John"; // zmienna lokalna lub globalna
```

Obiekt window – obiekt reprezentujący aktualny stan przeglądarki

WNIOSEK: Nigdy nie zapominaj o słówku var, deklarując zmienną

Zmienne - nazewnictwo

- Nazwy mogą posiadać cyfry, litery, podkreślniki lub znak dolara
- Nazwy muszą zaczynać się od litery lub podkreślnika
- Wielkość znaków ma znaczenia – JS jest *case sensitive*
- Słowa zarezerwowane nie mogą być użyte jako nazwy
- Używamy camelCase
- Wyłącznie język angielski

„There are only two hard things in Computer Science: cache invalidation and naming things.”
-- Phil Karlton

typeof() – sprawdzanie typu zmiennej

```
var name = "Janusz";
typeof(name); // string
typeof name; // string
typeof(5); // number
typeof 4; // number
Typeof null // object
```

Zarezerwowane słowa kluczowe

Keywords

Reserved keywords as of ECMAScript 2015

- `break`
- `case`
- `catch`
- `class`
- `const`
- `continue`
- `debugger`
- `default`
- `delete`
- `do`
- `else`
- `export`
- `extends`
- `finally`
- `for`
- `function`
- `if`
- `import`
- `in`
- `instanceof`
- `new`
- `return`
- `super`
- `switch`
- `this`
- `throw`
- `try`
- `typeof`
- `var`
- `void`
- `while`
- `with`
- `yield`

ZADANIA

6.
JavaScript – Funkcje

FUNKCJA

Funkcja – (opcjonalnie parametryzowany) blok kodu zaplanowany do wykonania konkretnego zadania, np. zsumowania dwóch liczb

Funkcje – deklarowanie vs wywołanie

Function declaration

```
function sum(a, b) {
    return a + b;
}
```

Invocation (wywołanie)

```
sum(3, 3); // OUTPUT: 6
sum(1, 5); // OUTPUT: 6
sum(9,-3); // OUTPUT: 6
```

Function expression

```
var sum = function (a, b) {
    return a + b;
};
```

Function Declaration vs Function Expression

Function declaration

```
sum(3, 4); // OUTPUT: 7

function sum(a, b) {
    return a + b;
}

sum(3, 4); // OUTPUT: 7
```

Function expression

```
sum(3, 4); // ERROR - sum is not
defined

var sum = function (a, b) {
    return a + b;
};

sum(3, 4); // OUTPUT: 7
```

Preferujemy używanie function declaration, bo na dowolnej wysokości kodu możemy sięgać po taką funkcję

Parametry

- Funkcja może przyjmować dowolnie wiele parametrów, dowolnego typu (string, numer i) – im mniej parametrów tym lepiej
- Funkcja nie waliduje typów przyjętych parametrów (w przeciwieństwie do Javy np., gdzie deklaruje się typy parametrów)
- Funkcja nie sprawdza ilości przyjętych parametrów

```
function sum(a, b) {
    return a + b;
}

sum("Hey", 5); // OUTPUT:
"Hey5"
```

```
function sum(a, b) {
    return a + b;
}

sum(3, 4, "Hey!"); // OUTPUT: 7;
```

Kontekst funkcji (scope)

Zmienne zadeklarowane wewnątrz funkcji nie są widoczne globalnie.
Funkcja ma dostęp do zmiennych globalnych (kontekstu globalnego)

```
console.log(VAT); // ERROR: VAT is not defined
var factor = 1.9; // zmienna globalna

function getGross(nettoPrice) {
    var VAT = 1.23;
    console.log(VAT); // 1.23
    console.log(factor); // 1.9;
    return nettoPrice * VAT;
}
```

Nested scope

```
function showName(firstName, lastName) {
    var nameIntro = "Your name is ";

    function makeFullName() {
        return nameIntro + firstName + " " + lastName;
    }

    return makeFullName();
}
showName("Michael", "Jackson"); // Your name is Michael Jackson
```

Funkcje zagnieżdżone mają dostęp do kontekstu globalnego oraz do kontekstu funkcji nadrzędnych

Domknięcia - closures

Domknięcie jest podtrzymaniem kontekstu funkcji pomimo zakończenia jej pracy.

```
function outer() {
  var a = 1;
  return function () {
    console.log(a++);
  };
}

var fnc = outer();
fnc();
```

Return w funkcji

- Return zawsze kończy działanie funkcji
- Wszystkie instrukcje po returnie nie zostaną wykonane
- Funkcja nie musi posiadać return (wtedy funkcja zwraca *undefined*)

```
function sum(a, b) {
    return a + b;
    console.log(„Hey!”); // UNREACHABLE CODE
}
```

```
function showGreeting() {
    console.log(„Hey!”);
}
```

Immediately-Invoked Function Expression (IIFE)

- Dzięki użyciu IIFE nie zaśmiecamy globalnej przestrzeni nazw

```
(function() {
…
})();
```

```
(function() {
    var a = b = 5;    // NIGDY TAK NIE RÓB!
})();
console.log(a);
console.log(b);
```

ZADANIA

JAVASCRIPT
Funkcje wbudowane

Built-in Functions – funkcje wbudowane

Oprócz funkcji, które możemy tworzyć sami, JavaScript udostępnia nam ogromny zestaw wbudowanych funkcji

Najbardziej popularne:

- **Math**: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math
- **Number**: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number
- **String**: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String
- **Array**: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Built-in Functions – funkcje wbudowane

Większość wbudowanych funkcji wywołujemy po kropce, bezpośrednio po zadeklarowanej wartości:

```
"Hej".toUpperCase();
(3.444).toFixed();
```

Lub przekazujemy wartość jako parametr

```
Math.round(3.4444);
```

Nadpisywanie wartości zmiennej

Większość metod nie zmienia wartości zmiennej, na której została wywołana:

```
var name = 'Tomek';

name.toUpperCase(); // string jest niemutowalny

console.log(name); // 'Tomek';
```

Nadpisywanie wartości zmiennej

Aby zmodyfikować wartość, z reguły przypisujemy jej aktualną wartość z wywołaniem danej metody

```
var name = 'Tomek’;

name = name.toUpperCase();

console.log(name); // 'TOMEK';
```

ZADANIA

7.
JavaScript – Operatory matematyczne

Operatory matematyczne

Podstawowe operatory matematyczne to:

+ (plus służy również do konkatenacji stringów)

- (odejmowanie)

* (mnożenie)

/ (dzielenie)

% (reszta z dzielenia, pomocna przy sprawdzaniu parzystości liczb)

++ (inkrementacja, najczęściej używana w pętlach)

-- (dekrementacja, najczęściej używana w pętlach)

Operatory przypisania

=	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

Inkrementacja: ++x vs x++

x++ najpierw egzekucja potem inkrementacja

++x najpierw inkrementacja potem egzekucja

```
var x = 1;
var y = x++; // y = 1, x = 2
var z = ++x; // z = 3, x = 3
```

ZADANIA

8.
Odczytywanie wartości obiektów i tablic

Odczytywanie wartości w obiektach i tablicach

```
var names = ['Janusz', 'Brajan', 'Seba', 'Mati'];
names[0] // 'Janusz', indeksy liczymy od 0
names[3] // 'Mati';

var person = {
    name: 'Mirek',
    surName: 'Nowak',
    profession: 'Handlarz samochodów'
};

person.name // 'Mirek'
person.surName // 'Nowak'
Person['name'] // 'Mirek'
```

ZADANIA

9.
JavaScript – Obiekt Date

Obiekt Date

Nowy obiekt Date tworzymy poprzez wywołanie new Date():

https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Date

```
var now = new Date();
console.log(now.getDay())
;
```

```
new Date("1995,11,17");
```

getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

Biblioteka MomentJS

Najczęściej używana bilblioteka do obsługi dat

https://momentjs.com/

ZADANIA

10. Hoisting

HOISTING

Hoisting – przenoszenie deklaracji zmiennych na początek kodu bez ich wartości

Hoisting

```
console.log(a); // "undefined" a nie "NOT DEFINED"
var a = 3;

// JAK KOD ZOSTANIE ZINTERPRETOWANY:
var a;
console.log(a);
a = 3;
```

Hoisting - również w funkcjach

```
function getGross(nettoPrice) {
    console.log(VAT); // undefined
    var VAT = 1.23;
    console.log(VAT);  // 1.23
    return nettoPrice * VAT;
}

// HOISTING
function getGross(nettoPrice) {
    var VAT;
    console.log(VAT); // undefined
    VAT = 1.23;
    console.log(VAT);  // 1.23
    return nettoPrice * VAT;
}
```

```
var one;
function getOne() {};
var two;
function getTwo() {};
var three;
getTwo();
getThree();
function getThree() {};
getOne();
```

```
(function() {
    var one;
    var two;
    var three;

    function getOne() {};
    function getTwo() {};
    function getThree() {};

    getOne();
    getTwo();
    getThree();
})();
```

Podsumowanie dnia I – podstawy JS

Co zapamiętać pod rozmowy kwalifikacyjne?

- Hoisting
- Closures
- Zmienne globalne vs lokalne
- Scope (zakres widoczności zmiennych)
- Co to jest IIFE i po co się używa
- Dlaczego JS’y podłącza się na koniec body w pliku HTML

Książki & Platformy & Blogi do nauki

JavaScript The Good Parts - O'Reilly
Eloquent JavaScript

https://www.codecademy.com/
https://www.codewars.com/

https://davidwalsh.name/
http://dailyjs.com/
https://www.sitepoint.com/javascript/
https://www.javascript.com/
https://www.polskifrontend.pl/ (agregator polskich blogów)

THE END

Pytania?

nastalytomasz@gmail.com
Tomasz Nastały @slack