



dr inż. Michał Malinowski
michal.malinowski@warszawa.merito.pl



[https://www.linkedin.com/in/
michał-malinowski-malkon/](https://www.linkedin.com/in/michał-malinowski-malkon/)

Podstawy JavaScript i Wprowadzenie do ES6

Wykład 01

JĘZYKI PROGRAMOWANIA
STUDIUM PRZYPADKU



- Zadanie: Wybranie tematu projektu aplikacji
- Wprowadzenie do JavaScript
- Narzędzia Deweloperskie
- Podstawy JavaScript
- Struktury Kontroli Przepływu
- Praca z Kolekcjami
- JSON - Standardowy Format Wymiany Danych
- Znaczenie ES6 dla Rozwoju aplikacji webowych
- ES6: Główne zmiany i ulepszenia

Zadanie: Wybranie tematu projektu aplikacji

Wybranie tematu projektu aplikacji

Temat wybrany indywidualnie przez studenta lub z zaproponowanej listy tematów zawartej w prezentacji.

Jako wynik należy przedstawić temat i jeden akapit tekstu opisujący temat.

Przedstawienie wyniku

- Dokument tekstowy PDF
- Umieszczenie wyniku w Moodle

Termin

- Podany przy zadaniu w Moodle (około 2 tygodnie)

Wprowadzenie do JavaScript

JavaScript

to wysokopoziomowy, interpretowany język programowania, który jest jednym z podstawowych technologii World Wide Web, obok HTML i CSS.

Został pierwotnie stworzony przez Brendana Eichę w 1995 roku, pracującego wtedy dla Netscape Communications Corporation. Początkowo był używany głównie do dodawania interaktywności, animacji i innych efektów w przeglądarkach internetowych.



Wprowadzenie do JavaScript

Kluczowe cechy JavaScript

Interpretowany i dynamiczny

Kod JavaScript jest wykonywany na bieżąco przez silnik JavaScript w przeglądarce lub środowisku uruchomieniowym, takim jak Node.js, bez potrzeby wcześniejszej kompilacji.

Obiektowy

JavaScript wspiera paradygmat programowania obiektowego, gdzie dane i funkcje są zorganizowane w obiekty.

Słabo typowany

W JavaScript nie musisz deklarować typów zmiennych; system typów jest dynamiczny, a typy zmiennych mogą się zmieniać w trakcie wykonania programu.

Wszechstronny

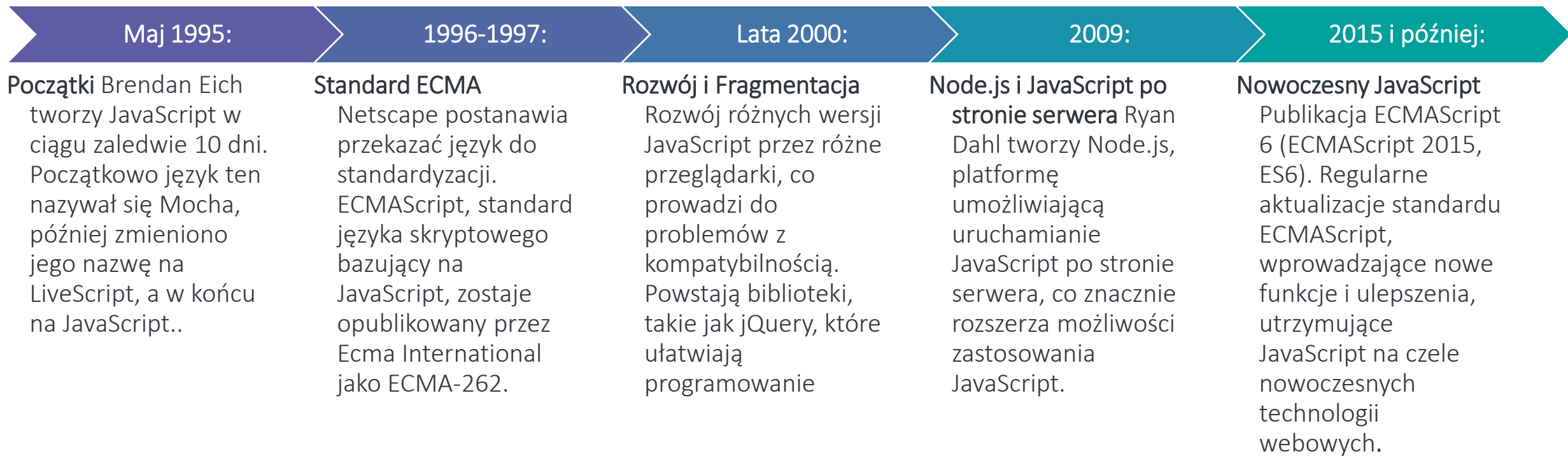
Początkowo używany tylko w przeglądarkach internetowych, JavaScript rozszerzył swoje zastosowanie na serwery (Node.js), aplikacje mobilne (React Native), gry, aplikacje desktopowe (Electron) i wiele innych.

Event-driven

JavaScript w przeglądarkach jest zorientowany na zdarzenia, co oznacza, że kod jest zwykle wykonywany w odpowiedzi na zdarzenia użytkownika, takie jak kliknięcia myszą, naciśnięcia klawiszy klawiatury itp.

Wprowadzenie do JavaScript

Historia



Narzędzia Deweloperskie

Podstawowe środowisko pracy

Przeglądarka internetowa:

Dostarcza silnik JavaScript, który interpretuje i wykonuje kod.

Wbudowane narzędzia deweloperskie (DevTools) do debugowania i testowania kodu.

Edytor tekstowy:

Może to być prosty edytor tak jak Notatnik, Notepad, Sublime Text, lub bardziej zaawansowane IDE jak Visual Studio Code, które oferuje dodatkowe funkcje przyjazne dla programistów, np. podświetlanie składni, autouzupełnianie i wtyczki rozszerzające funkcjonalność.

Narzędzia Deweloperskie

Zaawansowane środowisko pracy

Środowisko uruchomieniowe Node.js:

Umożliwia uruchamianie JavaScriptu poza przeglądarką, co jest potrzebne np. dla aplikacji serwerowych.

System kontroli wersji, np. Git:

Umożliwia zarządzanie kodem źródłowym i współpracę w zespole.

Menadżery pakietów, np. npm lub yarn:

Służą do zarządzania zależnościami i bibliotekami zewnętrznymi.

Narzędzia Deweloperskie

Narzędzia Przeglądarki (DevTools)

Zestaw narzędzi wbudowanych w przeglądarki internetowe, takie jak Chrome, Firefox, Safari, które pomagają w debugowaniu kodu, analizie wydajności, inspekcji elementów DOM, zarządzaniu siecią i przechowywaniem.

Kluczowe Funkcje

Inspektor
elementów i edytor
CSS.

Konsola JavaScript
dla testowania kodu
i wyświetlania
błędów.

Debugger do
zatrzymywania kodu
i analizy wartości
zmiennych.

Analiza wydajności i
użycia zasobów.

Narzędzia Deweloperskie

Środowiska Programistyczne (IDEs)

Zintegrowane środowisko programistyczne (IDE) to aplikacja zapewniająca kompleksowe udogodnienia do pisania, modyfikowania, testowania i debugowania kodu.

- Visual Studio Code: lekkie, rozszerzalne, dla różnych technologii.
- IntelliJ IDEA: potężne środowisko dla Javy i innych języków.
- Eclipse: używane głównie dla Javy, ale wspiera też inne języki.

Kluczowe Funkcje

Podświetlanie
składni,
autouzupełnianie
kodu, refaktoryzacja.

Wbudowane
narzędzia do kontroli
wersji i
debugowania.

Wsparcie dla wielu
wtyczek i rozszerzeń.

Narzędzia Deweloperskie

Systemy Kontroli Wersji

System kontroli wersji pozwala na śledzenie i zarządzanie zmianami w kodzie źródłowym, wspierając równoczesną pracę wielu deweloperów.

- Git: Rozproszony system kontroli wersji, niezwykle elastyczny w zarządzaniu wieloma gałęziami pracy równocześnie.
- Subversion (SVN): Centralizowany system kontroli wersji, zapewniający prostotę i łatwość w zarządzaniu liniowymi przepływami pracy.
- Mercurial: Rozproszony system kontroli wersji, podobny do Git, charakteryzujący się prostotą w obsłudze i szybkością działania.

Kluczowe Funkcje

Historia zmian z dokładnymi informacjami o autorach i zmianach.

Wsparcie dla pracy zespołowej i rozwiązywania konfliktów.

Integracja z wieloma środowiskami programistycznymi i platformami hostingowymi

Narzędzia Deweloperskie

Menadżery Pakietów

Są narzędziami służącymi do automatyzacji procesu instalacji, aktualizacji, konfiguracji i usuwania kodu bibliotek lub zależności w projektach programistycznych.

- npm (Node Package Manager): Standardowy menadżer pakietów dla środowiska Node.js i języka JavaScript, używany do zarządzania zależnościami projektów.
- Yarn: Alternatywny menadżer pakietów dla JavaScript, stworzony przez Facebooka, znany z szybkości i niezawodności.

Kluczowe Funkcje

Zarządzanie
Zależnościami

Ułatwienie
Współpracy w
Zespole

Automatyzacja
i Skrypty

Podstawy JavaScript

Deklaracja Zmiennych: var, let, const

var - zmienna o zasięgu funkcji, podlegająca hoistingowi.

let - zmienna o zasięgu blokowym, bez hoistingu.

const - stała o zasięgu blokowym, nie można jej ponownie przypisać.

Przykłady

```
var name = 'Jan';
```

```
let age = 25;
```

```
const PI = 3.14;
```

Hoisting (podnoszenie) w JavaScript

to zachowanie języka, polegające na tym, że deklaracje zmiennych (przy użyciu słowa kluczowego var) i funkcji są "przenoszone" na początek zakresu, w którym zostały zadeklarowane, przed wykonaniem kodu. Oznacza to, że można odwołać się do zmiennej lub funkcji przed jej faktyczną deklaracją w kodzie.

```
console.log(x); // undefined, a nie błąd odwołania  
var x = 5;  
console.log(x); // 5
```

Podstawy JavaScript

Funkcje: Deklaracje, Wyrażenia, Arrow Functions

Deklaracja funkcji: klasyczny sposób tworzenia funkcji.

Wyrażenie funkcji: przypisanie funkcji do zmiennej.

Arrow functions: skrócona forma wyrażen funkcji, nie posiada własnego this.

Przykłady

Deklaracja: `function powiedzWitaj() { console.log('Witaj!'); }`

Wyrażenie: `const powiedzCzesc = function() { console.log('Cześć!'); };`

Arrow function: `const dodaj = (a, b) => a + b;`

Struktury Kontroli Przepływu

Instrukcje Warunkowe

Służą do wykonania kodu w zależności od spełnienia określonego warunku.

if, else if, else - podstawowa struktura warunkowa.

Operator warunkowy (ternary operator): warunek ? wyrażenie1 : wyrażenie2.

```
for (let i = 0; i <= punkty; i++) {  
  if (i < 50) {  
    ocena = 'niedostateczny';  
  } else if (i < 70) {  
    ocena = 'dostateczny';  
  } else if (i < 90) {  
    ocena = 'dobry';  
  } else {  
    ocena = 'bardzo dobry';  
  }  
}
```

```
ocena === 'bardzo dobry' ? console.log("Gratulacje! Twoja ocena to:", ocena) :  
                           console.log("Twoja ocena to:", ocena);
```

Struktury Kontroli Przepływu

Pętle

Umożliwiają wielokrotne wykonanie bloku kodu.

for - określa liczbę iteracji.

while, do...while - wykonuje kod, dopóki warunek jest spełniony.

for...in i for...of - iteracja przez właściwości obiektu lub elementy iterowalne.

```
for (let i = 0; i < 5; i++) {  
  console.log('Wartość i:', i);  
}
```

```
let x = 0;  
while (x < 5) {  
  console.log('Wartość x:', x);  
  x++;  
}
```

```
let y = 0;  
do {  
  console.log('Wartość y:', y);  
  y++;  
} while (y < 5);
```

```
const tablica = ['a', 'b', 'c'];  
for (let wartosc of tablica) {  
  console.log('Wartość:', wartosc);  
}
```

```
const obiekt = { a: 1, b: 2, c: 3 };  
for (let klucz in obiekt) {  
  console.log(`Klucz: ${klucz}, Wartość: ${obiekt[klucz]}`);  
}
```

Struktury Kontroli Przepływu

Obsługa Wyjątków

Pozwala na obsługę błędów w programie.

try - blok kodu, gdzie może wystąpić wyjątek.

catch - blok kodu, który jest wykonywany, gdy w bloku try wystąpi wyjątek.

finally - blok kodu, który zostanie wykonany po blokach try i catch, niezależnie od wyniku.

```
try {  
  // Kod, który może generować wyjątek  
  let wynik = divide(10, 0);  
  console.log(wynik);  
} catch (e) {  
  // Kod ten zostanie wykonany, jeśli w bloku `try` wystąpi błąd  
  console.error("Wystąpił błąd podczas dzielenia:", e);  
} finally {  
  // Ten kod zostanie wykonany niezależnie od wyniku  
  console.log("Działanie procedury dzielenia zakończone.");  
}
```

Praca z Kolekcjami

Tablice

to uporządkowana, listopodobna struktura danych, która może przechowywać elementy różnych typów, w tym liczby, stringi, obiekty, a nawet inne tablice.

Deklaracje tablicy

```
let mojaTablica = [];
```

```
let liczby = [1, 2, 3, 4, 5];  
let owoce = ['jabłko', 'banan', 'cytryna'];
```

```
let mojaInnaTablica = new Array();  
let kolory = new Array('czerwony', 'zielony', 'niebieski');
```

Praca z Kolekcjami

Cechy tablicy

Indeksowane od zera:

Pierwszy element tablicy ma indeks 0, drugi element ma indeks 1, i tak dalej.

Długość tablicy:

Właściwość `length` tablicy zwraca liczbę jej elementów.

Heterogeniczność:

Tablice mogą zawierać elementy różnych typów danych.

Dynamiczne rozmiary:

Tablice w JavaScript nie mają stałego rozmiaru i mogą być dynamicznie rozszerzane lub zmniejszane.

Praca z Kolekcjami

Metody dla tablic

.push() – dodaje elementy na końcu tablicy.

.pop() – usuwa ostatni element z tablicy.

.shift() – usuwa pierwszy element z tablicy.

.unshift() – dodaje elementy na początku tablicy.

.forEach() – wykonuje funkcję dla każdego elementu.

```
let owoce = ['jabłko', 'banan'];
owoce.push('pomarańcza');
console.log(owoce); // ["jabłko", "banan", "pomarańcza"]
```

```
let liczby = [1, 2, 3];
let usunietaLiczba = liczby.pop();
console.log(liczby); // [1, 2]
console.log(usunietaLiczba); // 3
```

```
let liczby = [1, 2, 3];
let usunietaLiczba = liczby.shift();
console.log(liczby); // [2, 3]
console.log(usunietaLiczba); // 1
```

```
let liczby = [2, 3, 4];
let nowaDlugosc = liczby.unshift(1);
console.log(liczby); // [1, 2, 3, 4]
console.log(nowaDlugosc); // 4
```

```
let liczby = [1, 2, 3];
liczby.forEach(function(liczba) {
  console.log(liczba);
});
```

Praca z Kolekcjami

Obiekty Map

Map to kolekcja par klucz-wartość.

Każdy klucz jest unikalny.

Pozwala na przechowywanie dowolnych typów danych jako klucze.

Metody dla Map

.set(key, value) – dodaje parę klucz-wartość do mapy.

.get(key) – zwraca wartość dla klucza.

.has(key) – sprawdza, czy klucz istnieje w mapie.

.delete(key) – usuwa parę klucz-wartość na podstawie klucza.

```
let mapa = new Map();
mapa.set('klucz1', 'wartość1');
mapa.set('klucz2', 100);
console.log(mapa); // Map { 'klucz1' => 'wartość1', 'klucz2' => 100 }
```

```
let mapa = new Map();
mapa.set('klucz1', 'wartość1');
let wartosc = mapa.get('klucz1');
console.log(wartosc); // 'wartość1'
```

```
let mapa = new Map();
mapa.set('klucz1', 'wartość1');
console.log(mapa.has('klucz1')); // true
console.log(mapa.has('klucz2')); // false
```

```
let mapa = new Map();
mapa.set('klucz1', 'wartość1');
mapa.delete('klucz1');
console.log(mapa.has('klucz1')); // false
```

Praca z Kolekcjami

Obiekty Set

Set to kolekcja (zbiór) unikalnych wartości.

Automatycznie usuwa duplikaty.

Nie ma dostępu do elementów poprzez indeks.

Metody dla Map

.add(value) – dodaje wartość do zbioru, jeśli jeszcze nie istnieje.

.has(value) – sprawdza, czy wartość istnieje w zbiorze.

.delete(value) – usuwa wartość ze zbioru.

.clear() – usuwa wszystkie elementy ze zbioru.

```
let numery = new Set();
numery.add(1);
numery.add(2);
console.log(numery.has(1)); // true
console.log(numery.has(3)); // false
```

```
let numery = new Set();
numery.add(1);
numery.add(2);
numery.delete(1);
console.log(numery.has(1)); // false
console.log(numery); // Set { 2 }
```

```
let numery = new Set();
numery.add(1);
numery.add(2);
numery.add(1); // Duplikat, nie zostanie dodany
console.log(numery); // Set { 1, 2 }
```

```
let numery = new Set();
numery.add(1);
numery.add(2);
numery.clear();
console.log(numery); // Set {}
```

JSON - Standardowy Format Wymiany Danych

JSON (JavaScript Object Notation)

to lekki format wymiany danych, łatwy do odczytania i zapisywania dla ludzi oraz do parsowania i generowania dla maszyn.

Struktura:

Lekki format tekstowy, bazujący na składni obiektów JavaScript. Używa nawiasów klamrowych {} do reprezentacji obiektów (czyli zestawów par klucz-wartość) i nawiasów kwadratowych [] dla tablic.

Czytelność:

Bardzo czytelny zarówno dla ludzi, jak i maszyn.

Użycie:

Szeroko stosowany w aplikacjach internetowych dla wymiany danych między klientem a serwerem, a także jako format konfiguracyjny i w web APIs.

JSON - Standardowy Format Wymiany Danych

Budowa

```
{
  "imie": "Jan",
  "nazwisko": "Kowalski",
  "wiek": 30,
  "miasto": "Warszawa",
  "znajomi": ["Anna", "Paweł", "Ewa"],
  "adres": {
    "ulica": "Krakowska 123",
    "kod_pocztowy": "00-000",
    "kraj": "Polska"
  }
}
```

- Plik tekstowy z rozszerzeniem .json
- Dane są przedstawione w formie par "klucz"-wartość".
- Obiekty (zestawy danych) są otoczone klamrami {}.
- Klucze są zawsze w podwójnych cudzysłowach (np. "imie").
- Wartości mogą być typu:
 - String
 - Number
 - Object (obiekt)
 - Array (tablica)
 - Boolean
 - Null.
- Tablice są listami wartości otoczone kwadratowymi nawiasami [].

JSON - Standardowy Format Wymiany Danych

Praca z JSON w JavaScript

Parsowanie JSON: `JSON.parse()`

służy do konwersji danych w formacie JSON na obiekt JavaScript.

```
const jsonString = '{"imie":"Anna","nazwisko":"Nowak","wiek":28}';
const osoba = JSON.parse(jsonString);

console.log(osoba.imie); // Wyświetli "Anna"
console.log(osoba.wiek); // Wyświetli 28
```

Zamiana obiektu na JSON: `JSON.stringify()`

konwertuje obiekt JavaScript lub wartość na łańcuch tekstowy w formacie JSON

```
const osoba = {
  imie: "Anna",
  nazwisko: "Nowak",
  wiek: 28
};

const jsonString = JSON.stringify(osoba);

console.log(jsonString); // Wyświetli '{"imie":"Anna",'
```

Znaczenie ES6 dla Rozwoju aplikacji webowych

ECMAScript 2015 (ECMAScript 6, ES6)

to aktualizacja do standardu języka JavaScript, która została zatwierdzona i opublikowana w czerwcu 2015 roku przez organizację Ecma International. Była to pierwsza znacząca aktualizacja języka od czasu ES5, która została wydana w 2009 roku. ES6 wprowadził wiele nowych funkcji i ulepszeń mających na celu ułatwienie pisania złożonego i modułowego kodu JavaScript.

ECMA "European Computer Manufacturers Association"

to międzynarodowa, prywatna organizacja non-profit założona w 1961 roku, mająca na celu rozwijanie i publikowanie standardów w dziedzinie informatyki i telekomunikacji. Organizacja ta zmieniła swoją nazwę na "Ecma International" w 1994 roku, aby lepiej odzwierciedlić jej globalny zasięg i działalność.

Znaczenie ES6 dla Rozwoju aplikacji webowych

Umożliwia tworzenie bardziej interaktywnych i dynamicznych aplikacji webowych.

Integracja z nowoczesnymi frameworkami (React, Angular, Vue) wspiera rozwój SPA (Single Page Applications).

ES6 wspiera nowoczesne praktyki programistyczne, ułatwiając współpracę w zespołach i utrzymanie kodu.

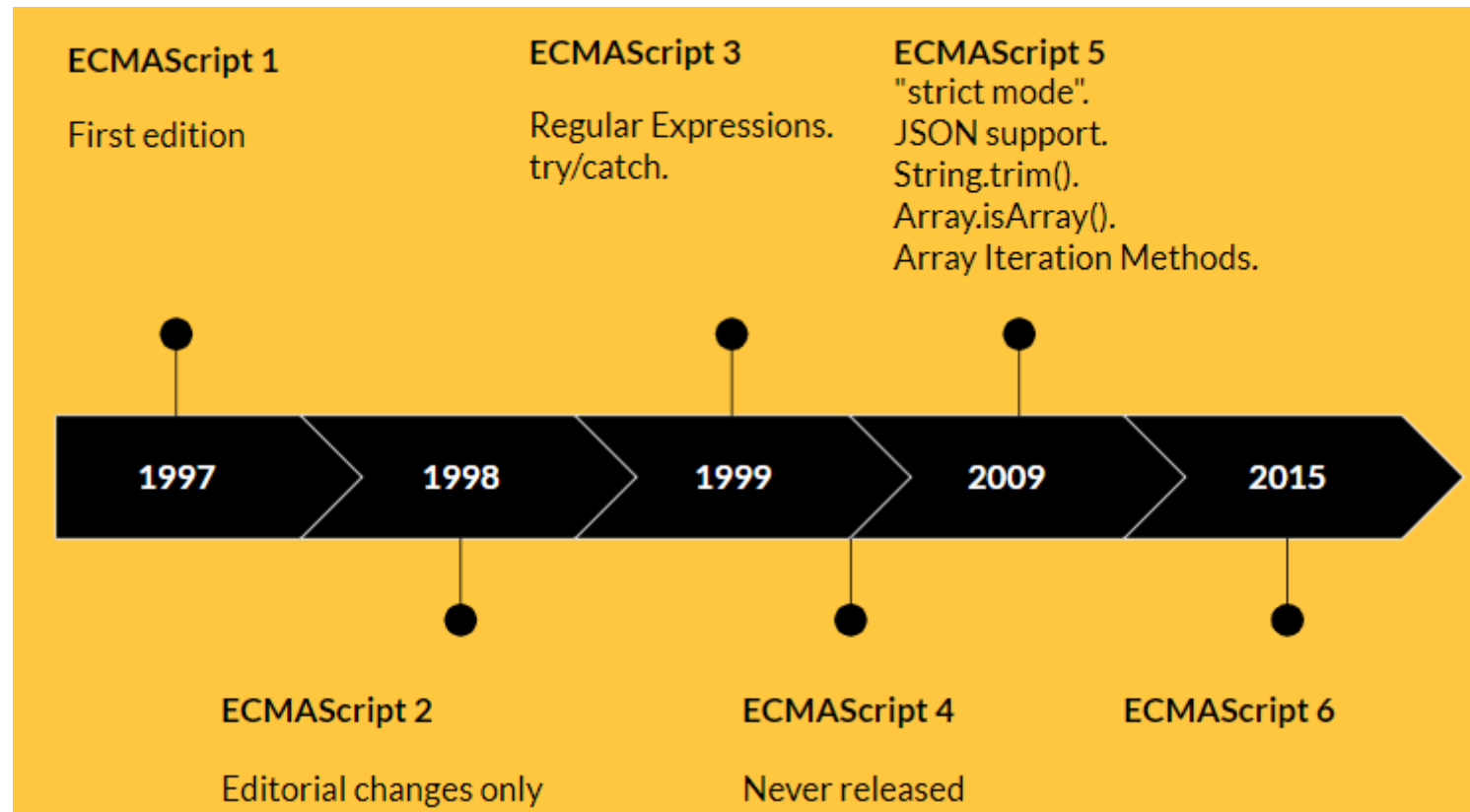
Klasy i Dziedziczenie: Ułatwiają programowanie obiektowe.

Wsparcie dla eksportu i importu, poprawa organizacji kodu.

Nowe Struktury Danych: Set i Map, ulepszenia dla lepszego zarządzania danymi.

Asynchroniczność: Promises i async/await dla bardziej intuicyjnego programowania asynchronicznego.

Znaczenie ES6 dla Rozwoju aplikacji webowych



ES6: Główne zmiany i ulepszenia

Główne zmiany i ulepszenia wprowadzone w ES6

Let i Const

Nowe słowa kluczowe do deklarowania zmiennych, oferujące lepszą kontrolę nad zasięgiem (scope) i unikanie problemów związanych z hoistingiem.

Klasy

Składnia klasy umożliwia bardziej zrozumiałe i uporządkowane tworzenie obiektów i dziedziczenie.

Moduły

Oficjalne wsparcie dla modułów umożliwia eksportowanie i importowanie wartości między różnymi plikami, co ułatwia budowanie modularnych aplikacji.

Arrow Functions

Funkcje strzałkowe (arrow functions) zapewniają skróconą składnię dla funkcji anonimowych i lepsze wiązanie kontekstu this.

Domyślne parametry funkcji

Możliwość ustawienia domyślnych wartości dla parametrów funkcji, jeśli nie zostaną one przekazane.

ES6: Główne zmiany i ulepszenia

Główne zmiany i ulepszenia wprowadzone w ES6

Template Literals

Uproszczona konkatenacja i wstawianie zmiennych do stringów za pomocą backticków (```) oraz wyrażeń `${}`.

Destructuring

Umożliwia łatwiejsze przypisywanie wartości z obiektów i tablic do zmiennych.

Promises

Obiekty promise umożliwiają łatwiejsze zarządzanie asynchronicznym kodem.

Enhanced Object Literals

Rozszerzona składnia literałów obiektów ułatwia deklarowanie obiektów.

Rest i Spread Operator

Umożliwiają pracę z nieokreśloną liczbą argumentów funkcji (rest) i rozszerzanie iterowalnych elementów, takich jak wyrażenia tablicowe (spread).

ES6: Główne zmiany i ulepszenia

let i const - Nowe sposoby deklaracji zmiennych

let - pozwala na deklarację zmiennych o zasięgu blokowym, unikając problemów z hoistingiem.

```
for(let i = 0; i < 5; i++) {  
  console.log(i);  
}  
// i nie jest dostępne poza pętlą
```

const - służy do deklarowania stałych, które nie mogą być ponownie przypisane.

```
const PI = 3.14;  
// PI = 3.15; // Błąd: ponowne przypisanie do stałej
```

ES6: Główne zmiany i ulepszenia

Arrow Functions - Zwięzła składnia funkcji

Funkcje strzałkowe oferują skróconą składnię i lepsze wiązanie this.

```
const add = (a, b) => a + b;  
console.log(add(2, 3)); // 5
```

Idealne dla callbacków i funkcji anonimowych.

```
[1, 2, 3].map(n => n * 2); // [2, 4, 6]
```

Callback (funkcja zwrotna)

to funkcja przekazana jako argument do innej funkcji, która ma być wywołana („callback”) w określonym momencie wykonania tej drugiej funkcji.

Funkcje anonimowe

to funkcja bez nazwy. Funkcje anonimowe są często używane jako argumenty podczas wywoływania funkcji wyższego rzędu, które przyjmują inne funkcje jako parametry, lub gdy funkcja jest używana tylko raz i nie ma potrzeby jej ponownego wywoływania.

ES6: Główne zmiany i ulepszenia

Template Literals - Ułatwione łączenie stringów

Umożliwiają wstawianie zmiennych i wyrażeń w stringi.

```
const name = "Marek";  
console.log(`Cześć, nazywam się ${name}!`); // Cześć, nazywam się Marek!
```

Domyślne Parametry Funkcji

Funkcje mogą mieć domyślne wartości parametrów.

```
function greet(name = "nieznajomy") {  
  console.log(`Cześć, ${name}!`);  
}  
greet(); // Cześć, nieznajomy!  
greet("Marek"); // Cześć, Marek!
```

ES6: Główne zmiany i ulepszenia

Rozprzestrzanie i Reszta Parametrów

Operator rozprzestrziania (...) do kopiowania elementów.

```
const arr1 = [1, 2];  
const arr2 = [...arr1, 3, 4]; // [1, 2, 3, 4]
```

Reszta parametrów (...args) do zbierania argumentów funkcji.

```
function sum(...numbers) {  
  return numbers.reduce((acc, num) => acc + num, 0);  
}  
console.log(sum(1, 2, 3)); // 6
```

NIE MOŻESZ
PRZESUNĄĆ OBRAZU PALCEM,
MUSISZ PODEJŚĆ DO
NASTĘPNEGO.

