www.szymkowiak.tychy.pl

Dobre praktyki pisania kodu w JavaScript

„[] opanowanie teorii jest zaledwie niezbędnym wstępem”

Grzegorz Szymkowiak

Dobre praktyki

- Czytelność kodu:
  - opisowe nazwy zmiennych, funkcji i klas, które odzwierciedlają ich znaczenie
  - wcięcia i spacje – kod staje się uporządkowany i łatwy do czytania

Dobre praktyki

- Konwencja camelCase – konwencja camelCase jest popularna i powszechnie akceptowana w społeczności JavaScript (stosowanie jednolitej konwencji nazewniczej ułatwia czytanie i zrozumienie kodu)

Dobre praktyki

- Konwencja camelCase :
  - przymiotnik po lewej stronie - premiumUser
  - is/are/has przy zmiennych true/false - isDarkMode, isPopupOpen
  - stałe zapisuj wielkimi literami - SECONDS = 60

Dobre praktyki

```
function numberToDigit(number) {

    while (number < 0) {
        console.log(number % 10);
        number = parseInt(number / 10);
    }

}
```

Dobre praktyki

```
function numberToDigit(number) {

    let digits = "";

    while (number < 0) {
        digits = number % 10;
        number = parseInt(number / 10);
    }

    return digits;
}
```

Dobre praktyki

- Używanie const i let zamiast var - const i let mają bardziej precyzyjne zakresy niż var, co pomaga uniknąć błędów związanych z hoistingiem, a kod jest bardziej przewidywalny

Dobre praktyki

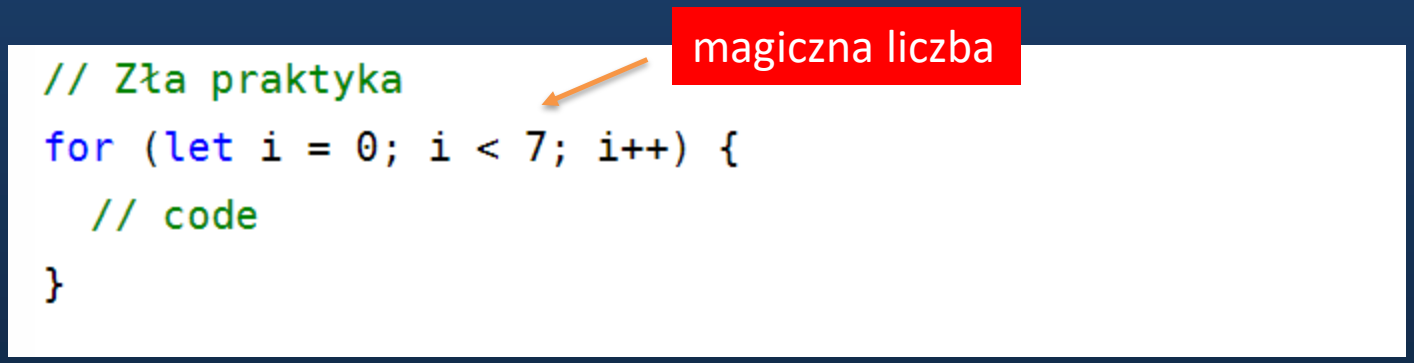
```
// Zła praktyka
let imie_uzytkownika = "John";

// Dobra praktyka
let imieUzytkownika = "John";
```

Dobre praktyki

- Unikanie globalnych zmiennych - zmienne globalne mogą prowadzić do konfliktów nazw i utrudniają zarządzanie kodem

Dobre praktyki

- Unikać „magicznych liczb” poprzez stosowanie stałych



Dobre praktyki

```
// Dobra praktyka
const DAYS_OF_THE_WEEK = 7;

for (let i = 0; i < DAYS_OF_THE_WEEK; i++) {
  // code
}
```

Dobre praktyki

- W warunkach związanych z porównywaniem należy użyć trzech znaków równości === zamiast dwóch ==

```
let x = 5;
let y = "5";

if(x === y) {
  console.log("Zmienne przechowują liczby o tej samej wartości");
}
```

Dobre praktyki

- Stosowanie arrow functions dla funkcji anonimowych - funkcje strzałkowe są bardziej zwięzłe, mają krótszą składnię oraz automatycznie wiążą wartość this, co ułatwia utrzymanie spójności kontekstu w funkcjach

Dobre praktyki

```
// Zła praktyka
let dodaj = function (a, b) {
  return a + b;
}

// Dobra praktyka
let dodaj = (a,b) => a + b;
```

Dobre praktyki

- Używanie modułów - moduły pomagają zorganizować kod, zapewniając lepszą separację funkcji i zmiennych, co ułatwia zarządzanie zależnościami, testowanie i ponowne użycie kodu

Dobre praktyki

- Obsługa błędów przy użyciu bloków try-catch - obsługa błędów pomaga uniknąć niekontrolowanego zakończenia działania programu i umożliwia bardziej precyzyjne reagowanie na błędy

Dobre praktyki

```
let a = 10, b = 0;

try {
  if(b === 0) {
    throw new Error('Dzielenie przez zero!');
  }

  let result = a / b;
  console.log("Wynik dzielenia", result);

} catch (error) {
  console.error('Wystąpił błąd:', error.message);
}
```

Dobre praktyki

- Komentarze tam, gdzie to konieczne - komentarze powinny być używane do tłumaczenia skomplikowanego kodu lub informowania o specyficznych rozwiązaniach

Dobre praktyki

```
// Zła praktyka
// Zmienna x
const x = 5;

// Dobra praktyka
const szerokoscOkna = 5;  // szerokość okna w pikselach
```

Dobre praktyki

- Testowanie kodu - testowanie kodu pomaga we wczesnym wykrywaniu błędów i utrzymaniu stabilności aplikacji (narzędzia do testowania: Jest, Mocha)

Dobre praktyki

- Testowanie kodu :
  - testy jednostkowe dla funkcji i klas
  - stosowanie techniki TDD (Test Driven Development)

Dobre praktyki

- Stosowanie narzędzi w środowisku developerskim:
  - ESLint – sprawdza składnię i semantykę kodu (używany do identyfikacji błędów)
  - Prettier – automatycznie formatuje kod
  - Live Server – umożliwia łatwe uruchomienie serwera HTTP

Korzyści z dobrych praktyki

Korzyści dobry praktyk

- Poprawa czytelności kodu
  ułatwia zrozumienie i utrzymanie kodu
- Łatwość wprowadzania zmian
  uproszczenie procesu modyfikacji i aktualizacji kodu
- Zmniejszenie objętości kodu
  skuteczniejsze i zwięzłe rozwiązania
- Praca zespołowa
  ułatwienie współpracy i dzielenia się kodem
- Uproszczenie testowania
  lepsza testowalność i szybsze wykrywanie błędów

www.szymkowiak.tychy.pl

KONIEC

Grzegorz Szymkowiak