

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ

Практична робота №1
з дисципліни
«Аналіз та рефакторинг коду»
на тему: «Code Convention»

Виконав:

ст. гр. ПЗП-22-5

Коноваленко Даніїл Миколайович

Перевірив:

ст. викладач каф. ПІ

Сокорчук Ігор Петрович

Харків 2024

ПРАВИЛА ОФОРМЛЕННЯ ПРОГРАМНОГО КОДУ

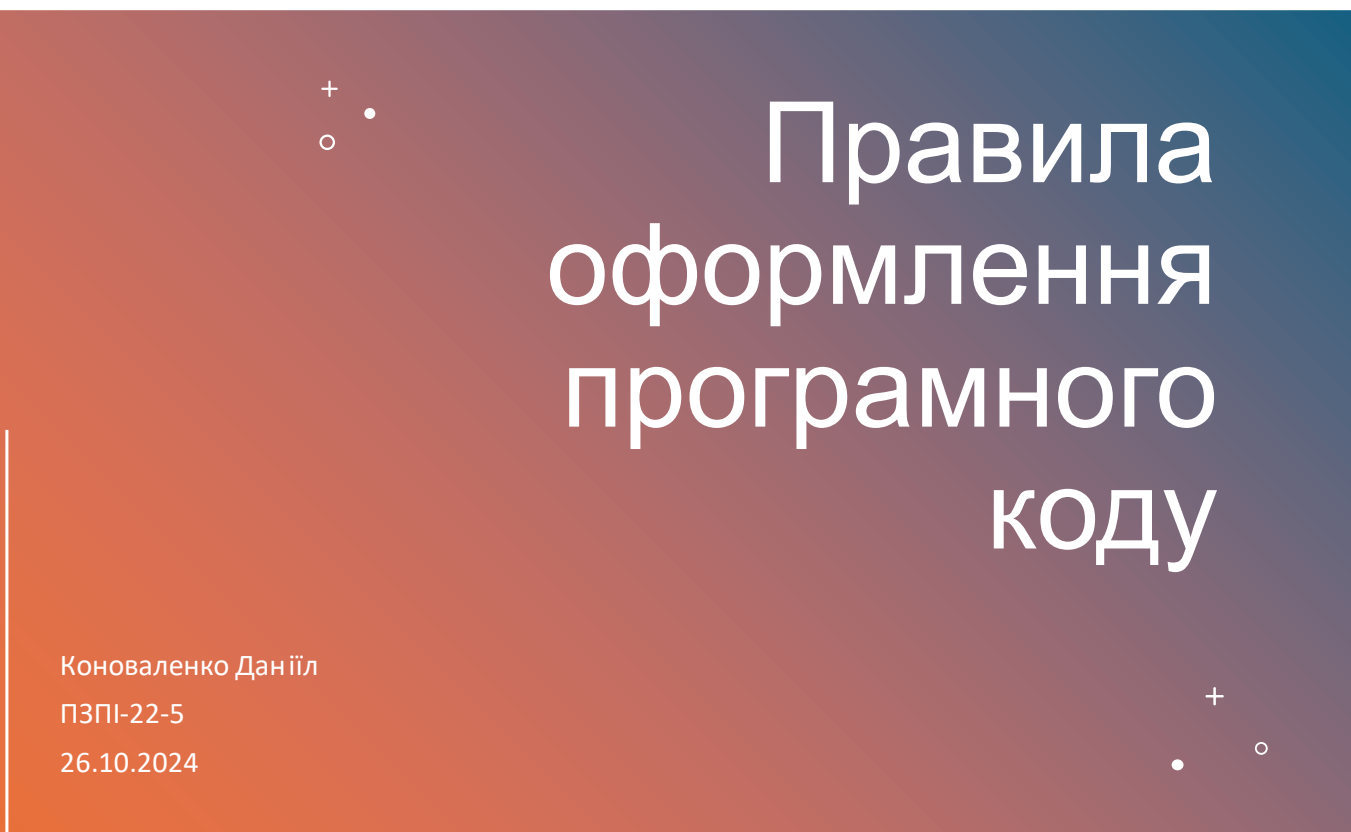
Мета роботи:

Ознайомитися з основними рекомендаціями щодо написання чистого, ефективного та підтримуваного коду для різних мов програмування, а також навчитися аналізувати та рефакторити код для покращення його якості.

Висновок:

Дотримання рекомендацій щодо написання чистого коду в JavaScript покращує читабельність, підтримку та оптимізацію коду, сприяє безпечнішій обробці помилок і відповідає парадигмам програмування. Ці рекомендації полегшують роботу з кодом, знижують ймовірність помилок та дозволяють зосередитися на продуктивній розробці.

Слайди презентації надані в Додатку А.



Правила оформлення програмного коду

Коноваленко Даніїл
ПЗПІ-22-5
26.10.2024



Вступ

Дотримання рекомендацій щодо написання чистого коду в JavaScript покращує читабельність, підтримку та оптимізацію коду, сприяє безпечнішій обробці помилок і відповідає парадигмам програмування.

Ці рекомендації полегшують роботу з кодом, знижують ймовірність помилок та дозволяють зосередитися на продуктивній розробці.

Стильові рекомендації

```

1  // Поганий приклад
2  function calculateSum(a,b){
3      console.log(a+b);}
4
5  // Гарний приклад
6  function calculateSum(a, b) {
7      console.log(a + b);
8  }
9  |

```

Відступи та пробіли

- Використовуйте однакові відступи у всьому проєкті.

Рекомендовано 2 або 4 пробіли для кожного рівня вкладення.

3

Стильові рекомендації

```

1  // Поганий приклад
2  let salutation = "Hello";
3  let name = 'World';
4
5  // Гарний приклад
6  let salutation1 = 'Hello';
7  let name1 = 'World';|

```

Консистентність у використанні лапок

- Виберіть між одинарними ' ' або подвійними " лапками та дотримуйтесь одного стилю.

4

Стильові рекомендації

```

1 // Поганий приклад
2 let message = 'Hello, ' + name + '!';
3
4 // Гарний приклад
5 let message1 = `Hello, ${name}!`;
6

```

Використання
шаблонних рядків
- Для об'єднання рядків
та змінних
використовуйте
шаблонні рядки з ``.

5

Правила найменування змінних, функцій та класів

```

1 // Поганий приклад
2 function calc(a, b) { return a + b; }
3
4 // Гарний приклад
5 function calculateSum(firstNumber, secondNumber) {
6   return firstNumber + secondNumber;
7 }
8

```

Змістовні імена
- Імена змінних і
функцій мають бути
зрозумілими та
відповідати змісту.

+
•
○

6

Правила найменування змінних, функцій та класів

```

1  // Поганий приклад
2  function Calculate_sum() { }
3  class user_account { }
4
5  // Гарний приклад
6  function calculateSum() { }
7  class UserAccount { }
8  |

```

Використовуйте camelCase для функцій і змінних
- camelCase підходить для іменування функцій та змінних, тоді як для назв класів використовуйте PascalCase.

+ •
○

7

Структура коду

```

1  // Поганий приклад
2  let firstNumber1 = 1;
3  let lastName1 = 'Deer';
4  let secondNumber1 = 2;
5  let firstName1 = 'John';
6
7  // Гарний приклад
8  let firstName = 'John';
9  let lastName = 'Deer';
10
11 let firstNumber = 1;
12 let secondNumber = 2;
13

```

Групуйте функції та змінні за їх призначенням
- Розміщуйте змінні, константи та функції, що мають схоже призначення, поруч для покращення читабельності коду.

+ •
○

8

Структура коду

```

1 // user.js
2 export class User { }
3 // main.js
4 import { User } from './user.js';
5

```

Розбивайте код на модулі

- Використовуйте модульну структуру, імпортуючи функції або класи в різні файли.

+
•
○

9

Принципи рефакторингу

```

1 // Поганий приклад
2 let total1 = price1 * 1.2;
3 let total2 = price2 * 1.2;
4
5 // Гарний приклад
6 function calculateTotal(price) {
7   |   return price * 1.2;
8 }
9

```

Уникайте повторення коду (DRY)

- Якщо певна логіка повторюється, винесіть її у функцію.

+
•
○

10

Оптимізація продуктивності

```
1 // Поганий приклад
2 for (let i = 0; i < array.length; i++) {
3   console.log(array.length);
4 }
5
6 // Гарний приклад
7 let length = array.length;
8 for (let i = 0; i < length; i++) {
9   console.log(length);
10 }
11 |
```

Уникайте зайвих викликів функцій у циклах

- Якщо функція не змінюється, викликайте її один раз до циклу.



11

Обробка помилок

```
1 async function fetchData() {
2   try {
3     let response = await fetch('https://api.example.com/data');
4     let data = await response.json();
5     return data;
6   } catch (error) {
7     console.error('Error fetching data:', error);
8   }
9 }
10 |
```

Використовуйте try-catch для обробки помилок

- Для роботи з можливими помилками в асинхронному коді використовуйте блоки try-catch.



12

Дотримання парадигм програмування

```

1  class Car {
2      constructor(make, model) {
3          this.make = make;
4          this.model = model;
5      }
6
7      displayInfo() {
8          return `${this.make} ${this.model}`;
9      }
10 }
11

```

Об'єктно-орієнтований підхід

- Використовуйте класи для структурованої роботи з об'єктами.

+
•
○

13

Документування коду

```

1  /**
2   * Обчислює суму двох чисел.
3   * @param {number} a - Перше число.
4   * @param {number} b - Друге число.
5   * @return {number} - Сума чисел.
6   */
7  function calculateSum(a, b) {
8      return a + b;
9  }
10

```

Документуйте функції та класи

- Використовуйте коментарі для пояснення логіки складних функцій та структур.

+
•
○

14

Висновки

Отже, основні принципи, які потрібно використовувати, це:

- Модульність;
- Оптимізація;
- Читабельність та стиль;
- Рефакторинг;
- Тестування;
- Коментування коду;
- Обробка помилок;
- Принцип DRY (Don't Repeat Yourself).

15

Джерела

- JavaScript Documentation
- Clean Code by Robert C. Martin
- JavaScript: The Good Parts by Douglas Crockford

16