

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХФРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра програмної інженерії

ЗВІТ

З практичної роботи № 1
з дисципліни «Аналіз та рефакторинг коду»
з теми: «Правила оформлення коду»

Виконала
ст. гр. ПЗПІ-22-7
Великотрав В. Ю.

Перевірів
ст. викладач кафедри ПІ
Сокорчук І. П.

Харків 2024

1.1 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.

2.1 Хід роботи

Обрати мову програмування для прикладів коду. Створити презентацію на тему «Правила оформлення програмного коду».

Було обрано мову програмування JavaScript. У презентації (Додаток Б) наведено основні рекомендації щодо оформлення програмного коду з описами.

Висновки

Набуто навичок написання чистого, ефективного та підтримуваного коду для різних мов програмування, а також навичок рефакторингу коду. Детально розглянуто основні правила оформлення коду.

Відео-презентація: <https://youtu.be/r7XjS8RJKzI>

Додаток А

Програмний код, використаний як приклад у презентації:

```
// Константи
const DISCOUNT_RATE = 0.1;

/**
 * Клас для роботи з користувачами
 */
class User {
  constructor(id, name, age) {
    this.id = id; // Унікальний ідентифікатор користувача
    this.name = name; // Ім'я користувача
    this.age = age; // Вік користувача
  }

  // Метод для привітання
  greet() {
    return `Hello, my name is ${this.name} and I am ${this.age} years old.`;
  }

  // Метод для отримання знижки
  calculateDiscount(total) {
    return total * DISCOUNT_RATE;
  }
}

/**
 * Функція для отримання даних користувачів (імітація API)
 * @returns {Promise<User[]>} Масив об'єктів користувачів
 */
async function fetchUsers() {
  // Імітуємо затримку, ніби ми отримуємо дані з API
  return new Promise(resolve => {
    setTimeout(() => {
      const mockData = [
        { id: 1, name: 'Alice', age: 25 },
        { id: 2, name: 'Bob', age: 30 },
        { id: 3, name: 'Charlie', age: 35 },
      ];
      resolve(mockData.map(user => new User(user.id, user.name, user.age)));
    }, 1000);
  });
}

/**
 * Функція для обчислення загальної суми покупок із знижкою
 * @param {number[]} prices Масив цін
 * @returns {number} Сума після знижки
 */
function calculateTotalWithDiscount(prices) {
```

```

    const total = prices.reduce((sum, price) => sum + price, 0); //
Функціональний підхід
    const discount = total * DISCOUNT_RATE;
    return total - discount;
}

/**
 * Основний виконуваний код
 */
(async function main() {
    console.log('Fetching users...');
    const users = await fetchUsers();

    if (users.length === 0) {
        console.warn('No users available to display.');
        return;
    }

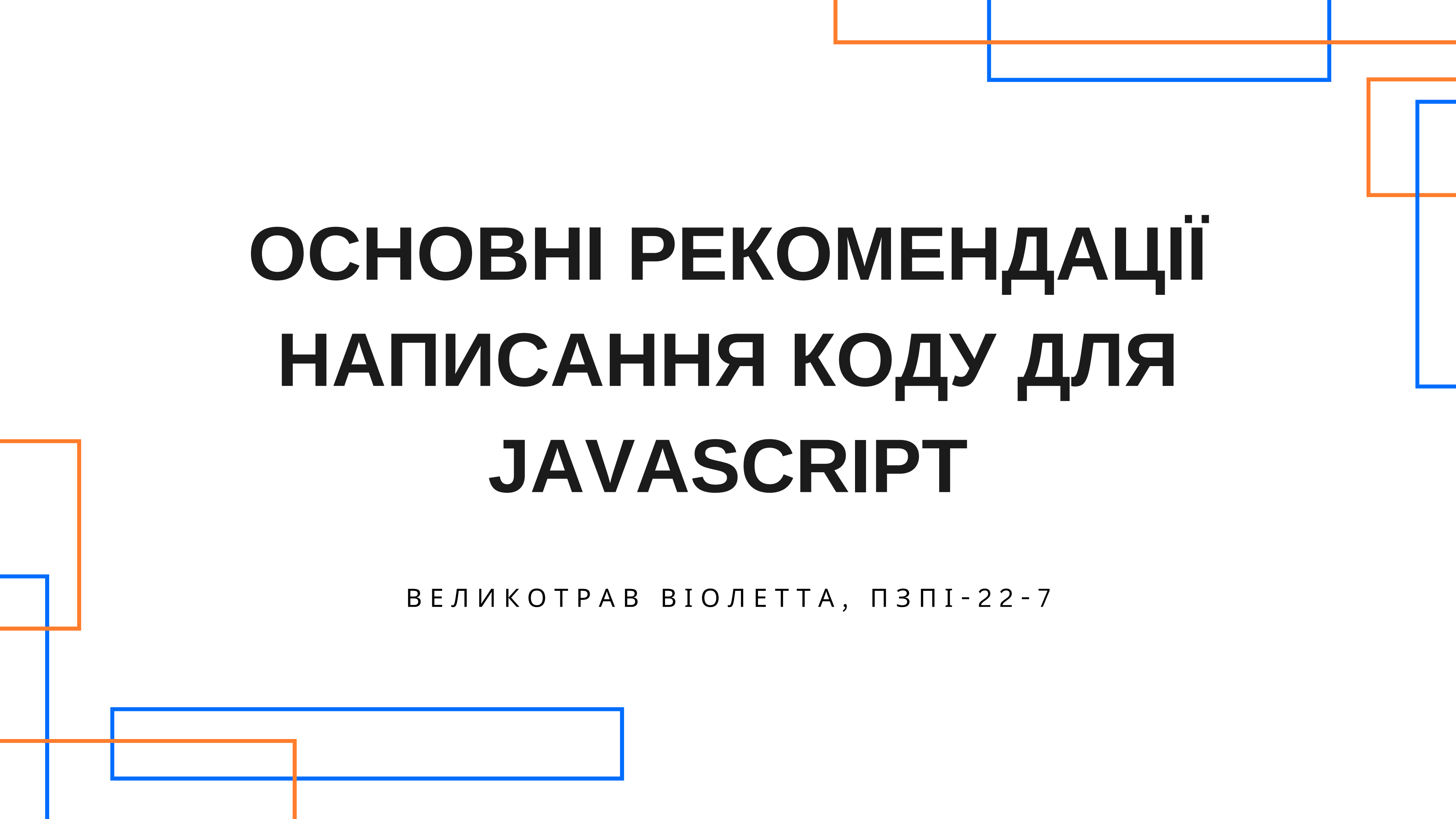
    // Демонстрація роботи з класами та функціями
    users.forEach(user => {
        console.log(user.greet()); // Виводимо привітання кожного
користувача
    });

    // Демонстрація роботи з масивами
    const prices = [100, 200, 300];
    const total = calculateTotalWithDiscount(prices);
    console.log(`Total after discount: ${total.toFixed(2)}`);
})();

```

Додаток Б

Презентація на тему «Правила оформлення програмного коду»

The background features several thin, overlapping rectangular outlines in orange and blue, creating a modern, abstract geometric pattern.

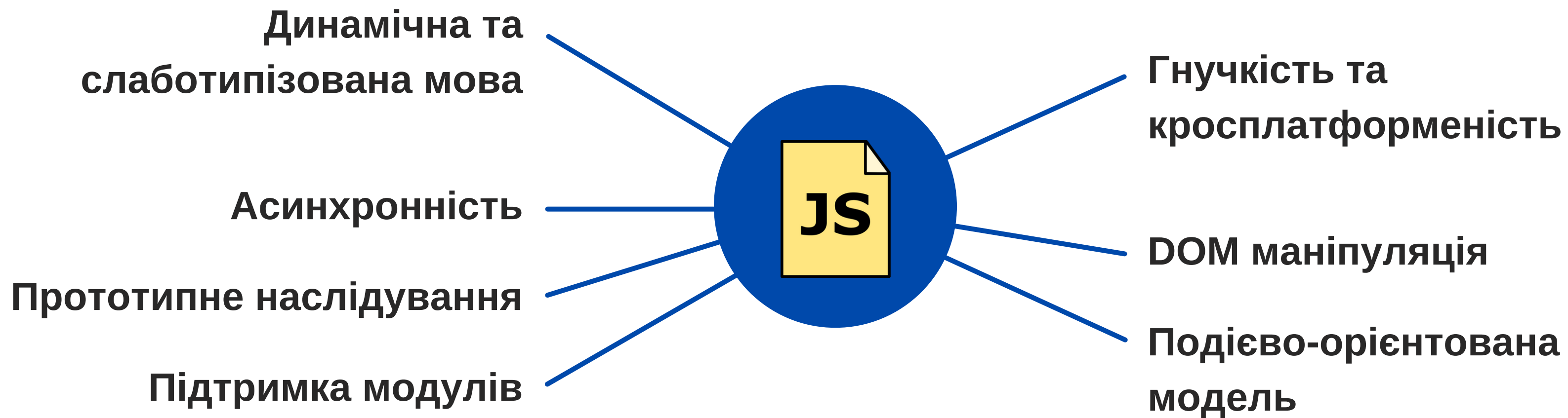
ОСНОВНІ РЕКОМЕНДАЦІЇ НАПИСАННЯ КОДУ ДЛЯ JAVASCRIPT

ВЕЛИКОТРАВ ВІОЛЕТТА, ПЗПІ-22-7

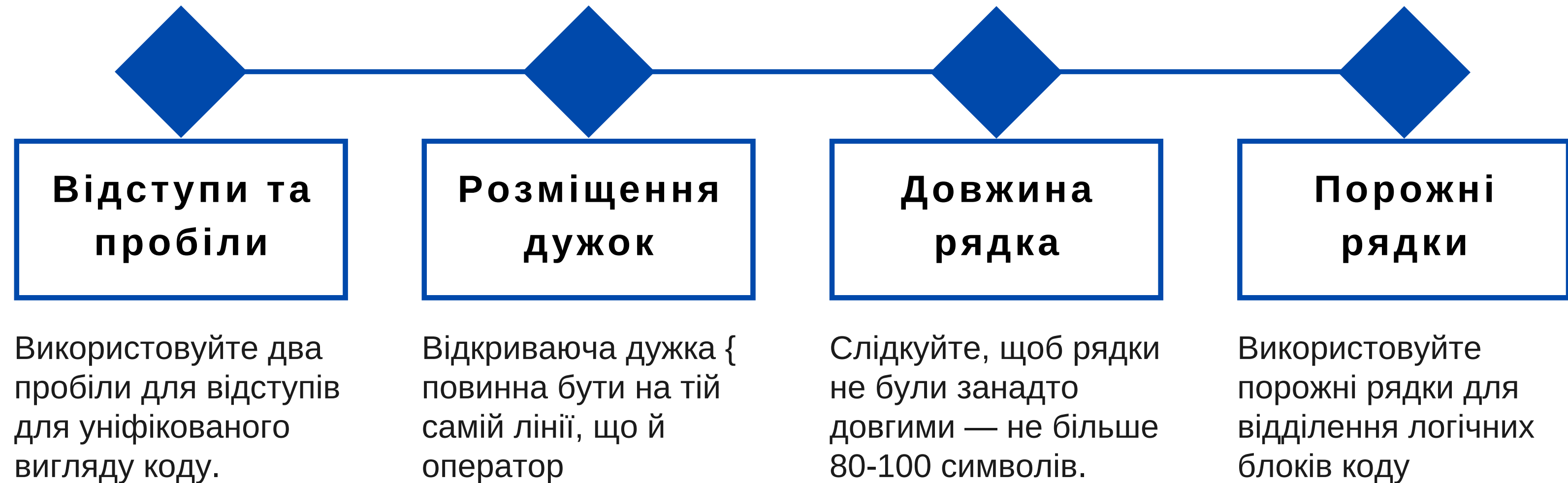
JAVASCRIPT

Це високорівнева, динамічна мова програмування, яка зазвичай використовується для створення інтерактивних веб-сайтів.

Вона була вперше розроблена в 1995 році компанією Netscape і стала однією з основних технологій веб-розробки поряд із HTML і CSS



СТИЛЬОВІ РЕКОМЕНДАЦІЇ



ПРИКЛАД КОДУ

```
// Поганий приклад
function calculateTotalWithDiscount(prices) {
  const total = prices.reduce((sum, price) => sum + price, 0); const discount = total * DISCOUNT_RATE; return total - discount;}

// Гарний приклад
function calculateTotalWithDiscount(prices) {
  const total = prices.reduce((sum, price) => sum + price, 0);
  const discount = total * DISCOUNT_RATE;
  return total - discount;
}
```

ПРАВИЛА НАЙМЕНУВАННЯ ЗМІННИХ, ФУНКЦІЙ ТА КЛАСІВ

Правильне найменування змінних, функцій та класів має вирішальне значення для зручності читання і розуміння коду.

Хороші назви допомагають іншим розробникам швидко зрозуміти призначення змінних і функцій без необхідності додаткових пояснень

ЗМІСТОВНІ ІМЕНА

Імена змінних і функцій мають бути описовими і чітко вказувати на їхнє призначення.

```
// Погана назва
a() {
  return `Hello, my name is ${this.name} and I am ${this.age} years old.`;
}

// Гарна назва
greet() {
  return `Hello, my name is ${this.name} and I am ${this.age} years old.`;
}
```

CAMELCASE ДЛЯ ЗМІННИХ І ФУНКЦІЙ

Імена змінних і функцій мають починатися з малої літери, а кожне наступне слово — з великої.

```
function calculateTotalWithDiscount(prices) {  
  const total = prices.reduce((sum, price) => sum + price, 0);  
  const discount = total * DISCOUNT_RATE;  
  return total - discount;  
}
```

PASCALCASE ДЛЯ КЛАСІВ

Імена класів починаються з великої літери, а кожне наступне слово також пишеться з великої.

```
class User {  
    constructor(id, name, age) {  
        this.id = id;  
        this.name = name;  
        this.age = age;  
    }  
}
```

УНИКАЙТЕ СКОРОЧЕНЬ ТА АБРЕВІАТУР

Назви мають бути інтуїтивно зрозумілими і не містити непотрібних скорочень або аббревіатур, які важко зрозуміти.

```
// Погана назва
clcltDisc(total) {
    return total * DISCOUNT_RATE;
}

// Гарна назва
calculateDiscount(total) {
    return total * DISCOUNT_RATE;
}
```

ВИКОРИСТОВУЙТЕ ДІЄСЛОВА ДЛЯ НАЗВ ФУНКЦІЙ

Ім'я функції має відображати дію, яку вона виконує.

Використовуйте дієслова на початку, наприклад get, set, calculate, fetch.

```
greet() {  
  return `Hello, my name is ${this.name} and I am ${this.age} years old.`;  
}  
  
calculateDiscount(total) {  
  return total * DISCOUNT_RATE;  
}
```

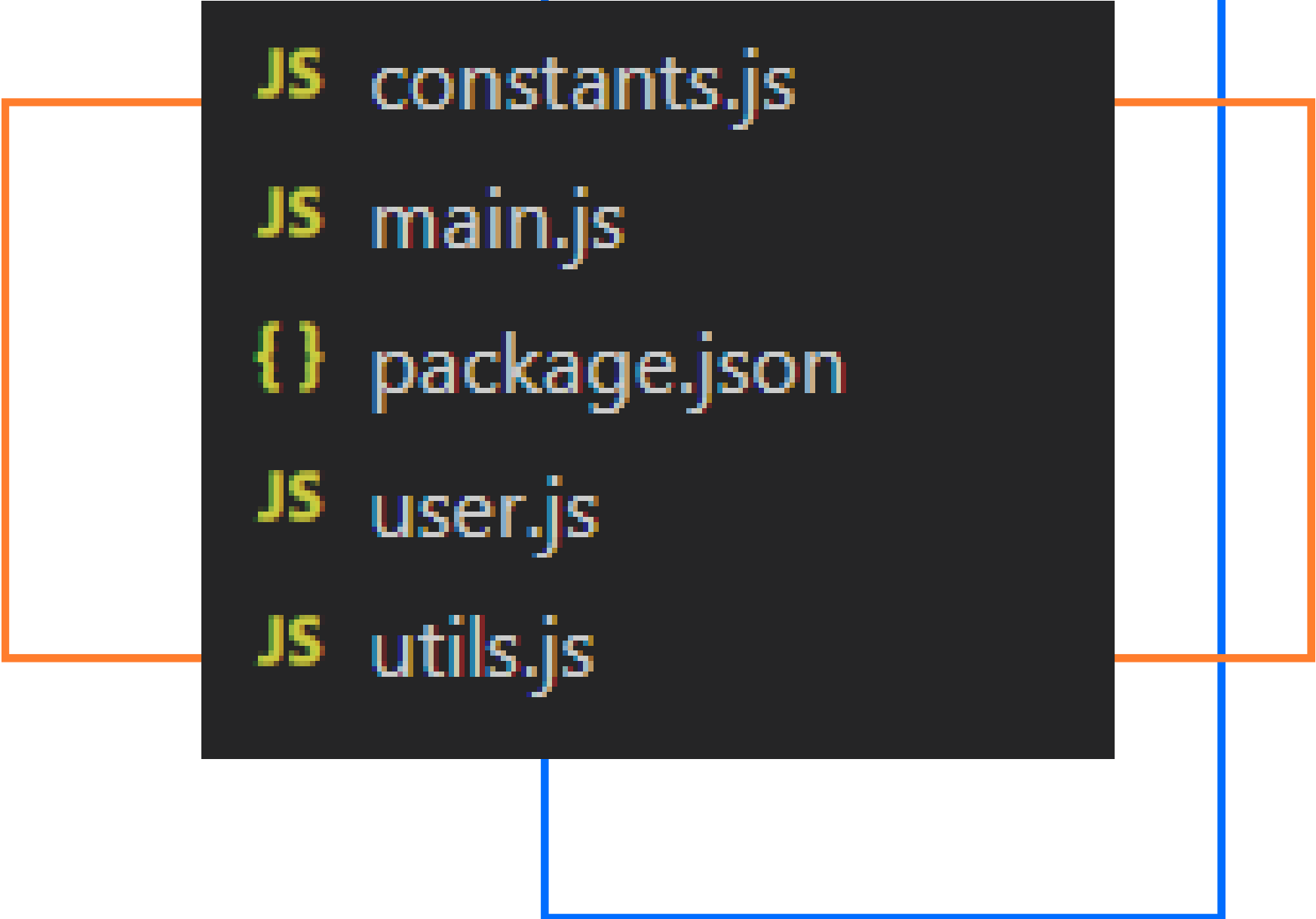

СТРУКТУРА КОДУ

Структура коду — це організація і розподіл коду в межах файлу або проекту.

Вона включає логічне розміщення функцій, класів, змінних, модулів та іншого функціоналу для забезпечення читабельності та простоти підтримки коду.

РОЗДІЛЯЙТЕ КОД НА МОДУЛІ

Використовуйте модулі для розділення логіки на окремі файли за функціональністю



```
JS constants.js  
JS main.js  
{ } package.json  
JS user.js  
JS utils.js
```

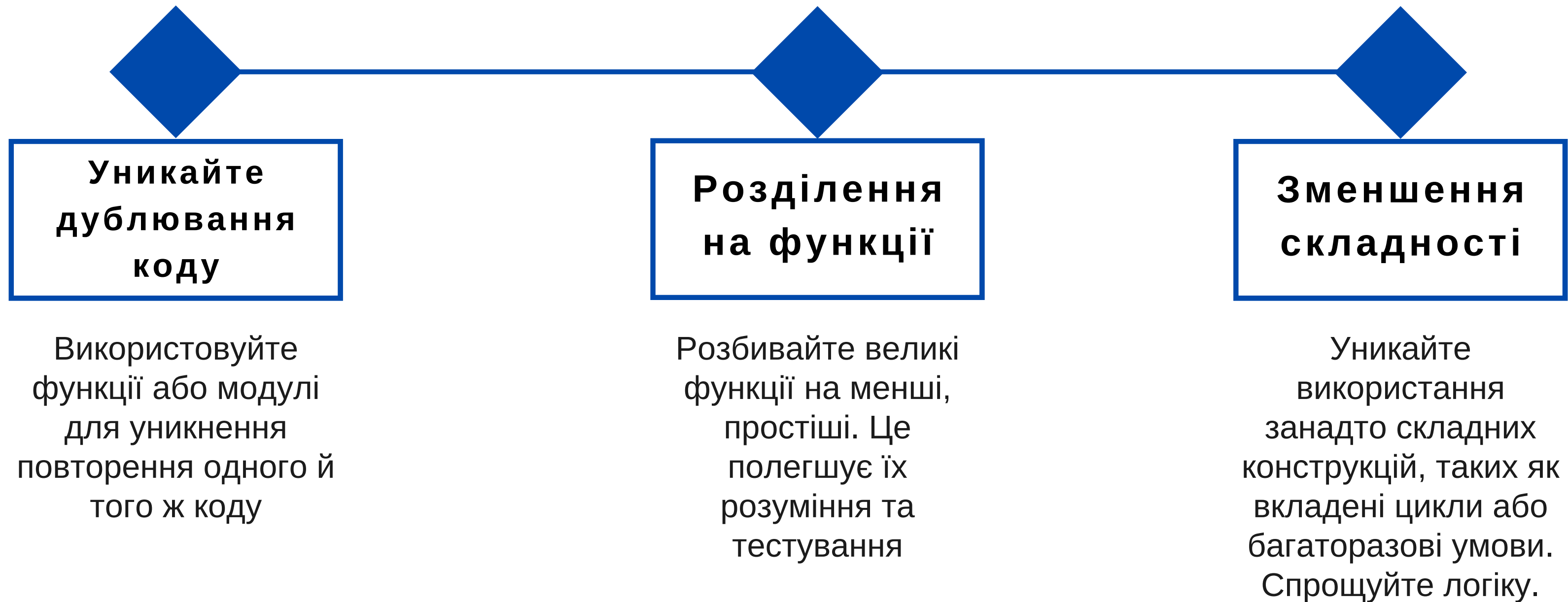
ЛОГІЧНО ГРУПУЙТЕ ФУНКЦІОНАЛЬНІ БЛОКИ

Розділяйте код на логічні секції за допомогою порожніх рядків. Наприклад, групуйте функції за їхньою метою або частини коду, що належать до певної задачі.

```
/**
 * Імітація API для отримання даних користувачів
 * @returns {Promise<User[]>} Масив користувачів
 */
export async function fetchUsers() {
  return new Promise((resolve) => {
    setTimeout(() => {
      const mockData = [
        { id: 1, name: "Alice", age: 25 },
        { id: 2, name: "Bob", age: 30 },
        { id: 3, name: "Charlie", age: 35 },
      ];
      resolve(mockData.map((user) => new User(user.id, user.name, user.age)));
    }, 1000);
  });
}

/**
 * Розрахунок загальної суми після знижки
 * @param {number[]} prices Масив цін
 * @returns {number} Сума після знижки
 */
export function calculateTotalWithDiscount(prices) {
  const total = prices.reduce((sum, price) => sum + price, 0);
  const discount = total * DISCOUNT_RATE;
  return total - discount;
}
```

ПРИНЦИПИ РЕФАКТОРИНГУ



ПРИКЛАД КОДУ

```
import { fetchUsers, calculateTotalWithDiscount } from "../utils.js";

(async function main() {
  console.log("Fetching users...");
  const users = await fetchUsers();

  if (users.length === 0) {
    console.warn("No users available to display.");
    return;
  }

  // Виведення привітань від користувачів
  users.forEach((user) => {
    console.log(user.greet());
  });

  // Обробка масиву цін
  const prices = [100, 200, 300];
  const total = calculateTotalWithDiscount(prices);
  console.log(`Total after discount: ${total.toFixed(2)}`);
})();
```

ОПТИМІЗАЦІЯ ПРОДУКТИВНОСТІ

Оптимізація продуктивності полягає в підвищенні швидкості та ефективності коду. Це може включати зменшення часу виконання, споживання пам'яті та загальної затримки.

ВИКОРИСТАННЯ ЛОКАЛЬНИХ ЗМІННИХ

Локальні змінні працюють швидше, ніж глобальні, оскільки доступ до них є більш ефективним.

```
// Погано
var globalVar = 10;

function calculate() {
    return globalVar * 2;
}

// Добре
function calculate(globalVar) {
    return globalVar * 2;
}

console.log(calculate(10));
```

ОПТИМІЗАЦІЯ ЦИКЛІВ

Уникайте повторних
обчислень у циклах,
наприклад, зберігайте
довжину масиву в змінній

```
// Погано
for (let i = 0; i < array.length; i++) {
  console.log(array[i]);
}

// Добре
const length = array.length;
for (let i = 0; i < length; i++) {
  console.log(array[i]);
}
```


УНИКНЕННЯ ПОВТОРНИХ ОБЧИСЛЕНЬ

Зберігайте результати
обчислень, якщо їх потрібно
використовувати кілька разів.

```
// Погано
function calculateSum(arr) {
  return arr.reduce((acc, val) => acc + val * Math.sin(val), 0);
}

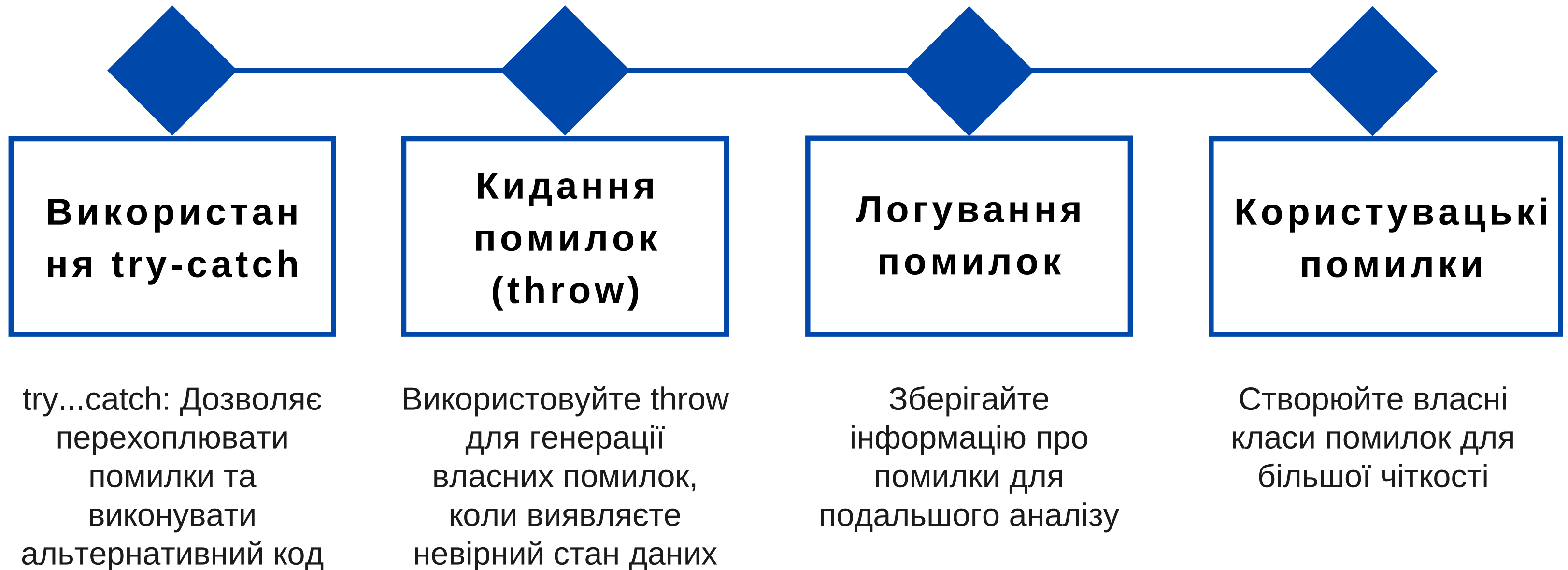
// Добре
function calculateSum(arr) {
  const sinValues = arr.map(Math.sin);
  return arr.reduce((acc, val, index) => acc + val * sinValues[index], 0);
}
```

ВИКОРИСТАННЯ МЕТОДІВ МАСИВУ

Використовуйте методи масивів (map, filter, reduce) замість ручних циклів для кращої читабельності та продуктивності.

```
export function calculateTotalWithDiscount(prices) {  
  const total = prices.reduce((sum, price) => sum + price, 0);  
  const discount = total * DISCOUNT_RATE;  
  return total - discount;  
}
```

ОБРОБКА ПОМИЛОК



ПРИКЛАД КОДУ

```
class CustomError extends Error {
  constructor(message) {
    super(message);
    this.name = "CustomError";
  }
}

const processNumber = (num) => {
  if (typeof num !== "number") {
    throw new CustomError("Input must be a number");
  }
  return num * 2;
};

const main = () => {
  const numbers = [1, "two", 3];

  numbers.forEach((num) => {
    try {
      const result = processNumber(num);
      console.log(`Processed number: ${result}`);
    } catch (error) {
      console.error(`${error.name}: ${error.message}`);
    }
  });
};
```

ДОТРИМАННЯ ПАРАДИГМ ПРОГРАМУВАННЯ

Це важливий аспект розробки програмного забезпечення, оскільки це дозволяє створювати структурований, модульний і підтримуваний код.

У JavaScript підтримуються кілька парадигм, включаючи об'єктно-орієнтоване програмування (ООП) і функціональне програмування (ФП).

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

```
class Person {  
  constructor(name, age) {  
    this.name = name;  
    this.age = age;  
  }  
  
  greet() {  
    console.log(  
      `Hello, my name is ${this.name} and I am ${this.age} years old.`  
    );  
  }  
}  
  
const person1 = new Person("Alice", 30);  
person1.greet();
```

ФУНКЦІОНАЛЬНЕ ПРОГРАМУВАННЯ

```
const square = (x) => x * x;  
  
const map = (arr, func) => arr.map(func);  
  
const numbers = [1, 2, 3, 4];  
const squares = map(numbers, square);  
console.log(squares);
```

ПАРАДИГМА НА ОСНОВІ ПОДІЙ

```
document.getElementById("myButton").addEventListener("click", function () {  
    alert("Button clicked!");  
});
```

У JavaScript широко використовується подієва модель, де функції реагують на події, такі як натискання кнопок або завантаження сторінок.

ТЕСТУВАННЯ ТА ДОКУМЕНТУВАННЯ

Тестування та документування коду є ключовими аспектами розробки, які забезпечують якість та підтримуваність програмного забезпечення.

Тестування допомагає виявити помилки та забезпечити, щоб код працював як очікується, тоді як документування забезпечує зрозумілість коду для інших розробників.

UNIT-ТЕСТИ

перевіряють окремі компоненти вашого коду (функції, класи тощо) на правильність роботи. Вони дозволяють ізолювати окремі частини коду для тестування.

```
function add(a, b) {  
  return a + b;  
}  
  
test("add should work correctly", () => {  
  expect(add(2, 3)).toBe(5);  
});
```

Integration-тести

перевіряють взаємодію між різними компонентами системи та їхню спільну роботу.

End-to-End тести

це форма тестування, яка перевіряє працездатність програмного продукту від початку до кінця, відтворюючи реальні сценарії використання та перевіряючи взаємодію між різними компонентами системи.

КОМЕНТАРІ

Додавайте коментарі до коду, щоб пояснити, що робить кожен блок коду, особливо у складних частинах

```
// Виведення привітань від користувачів
users.forEach((user) => {
  console.log(user.greet());
});

// Обробка масиву цін
const prices = [100, 200, 300];
const total = calculateTotalWithDiscount(prices);
console.log(`Total after discount: ${total.toFixed(2)}`);
})();
```

ДОКУМЕНТАЦІЯ ЗА ДОПОМОГОЮ JSDOC

Використовуйте JSDoc для генерації документації, яка пояснює функції, параметри та повернені значення.

```
/**
 * Розрахунок загальної суми після знижки
 * @param {number[]} prices Масив цін
 * @returns {number} Сума після знижки
 */
export function calculateTotalWithDiscount(prices) {
  const total = prices.reduce((sum, price) => sum + price, 0);
  const discount = total * DISCOUNT_RATE;
  return total - discount;
}
```

ДОКУМЕНТУВАННЯ API

Якщо ви створюєте API,
документуйте його за
допомогою OpenAPI або
Swagger

```
description: Unauthorized error
content:
  application/json:
    schema:
      type: object
      required:
        - status
        - message
        - data
      properties:
        status:
          type: integer
          example: 401
        message:
          type: string
          example: UnauthorizedError
        data:
          type: object
          required:
            - message
          properties:
            message:
              type: string
              example: 'Access token expired'
```

ВИСНОВКИ

Запропоновані рекомендації підвищують зрозумілість, надійність і підтримуваність коду, що дозволяє легше працювати над його вдосконаленням та масштабуванням. Тестування, документування та дотримання парадигм допомагають швидко знаходити помилки, спрощують внесення змін та забезпечують стабільність програми в довгостроковій перспективі.

СПИСОК ДЖЕРЕЛ

- ◆ Airbnb Javascript Style Guide (<https://github.com/airbnb/javascript?tab=readme-ov-file#the-javascript-style-guide-guide>)
- ◆ Google JavaScript Style Guide (<https://google.github.io/styleguide/jsguide.html>)
- ◆ Refactoring: Improving the Design of Existing Code. AddisonWesley, 2018.