

Основи роботи з функціями

а́дження

01 / Що таке функція та її структура

02 / Види оголошення функцій

03 / Рекурсія

Функції

Функція - це блок коду, призначений для виконання певної задачі.

Щоб не повторювати один і той же самий код у багатьох місцях, придумані функції.

Функції є основними "будівельними блоками" програми.

Оголошення функції

Функція визначається за допомогою ключового слова **function**, за яким слідує назва функції, дужки () та тіло функції, узяті в фігурні дужки { }.

```
function sayHello() {  
    console.log("Hello, world!");  
}  
  
sayHello(); // виклик функції
```

Аргументи / параметри

Ми можемо передати всередину функції будь-яку інформацію, використовуючи параметри.



```
function sayHello(name) {  
  console.log(`Hello, ${name}!`);  
}  
  
sayHello("Serhii");  
sayHello("Bob");
```

Повернення значення

Функція може повернути результат, який буде переданий у код, що її викликав. Для цього використовується ключове слово `return`

```
function add(a, b) {  
    return a + b;  
}  
  
let result = add(5, 3); // 8  
let otherResult = add(15, 10); // 25
```

Параметри за замовчуванням

Параметри за замовчуванням у функціях дозволяють встановити значення за замовчуванням для одного або кількох параметрів, якщо вони не надаються при виклику функції.

Це особливо корисно, коли деякі параметри функції не є обов'язковими та можуть мати розумне значення за замовчуванням.

Розглянемо на прикладі.

Задача

Написати функцію `roundNumber`, яка буде округлювати число до переданої кількості знаків після коми і повертати результат у вигляді числа

```
function roundNumber(..) {  
    // ...  
}  
  
console.log(roundNumber(20.333333, 2)); // 20.33  
console.log(roundNumber(1.151515, 3)); // 1.151
```


Правильне іменування функцій

Функція – це дія. Тому ім'я функції зазвичай є дієсловом.

Воно має бути простим, точним і описувати дію функції, щоб програміст, який читатиме код, отримав правильне уявлення про те, що робить функція.

Розглянемо на прикладах.

Гарні імена

```
// Із цієї назви зрозуміло, що функція дістає дані користувача.  
function getUserData() {}  
  
// Зрозуміло, що функція обчислює загальну вартість.  
function calculateTotalPrice() {}  
  
// Назва сигналізує, що функція перевіряє, чи є адреса електронної пошти коректною.  
function isValidEmail() {}  
  
// Із імені можна зрозуміти, що функція конвертує валюту.  
function convertCurrency() {}  
  
// // Зрозуміло, що функція створює нового користувача  
function createNewUser() {}
```

Погані імена

```
// Це абсолютно неінформативно. З імені незрозуміло, що саме робить ця функція.  
function a() {}
```

```
// Використання чисел у назвах функцій  
function fn1() {}
```

```
// Дуже загальна назва, яка не допомагає зрозуміти, що саме робить функція.  
function doStuff() {}
```

```
// Скорочення. Знову ж таки, занадто неінформативно.  
function pC() {}
```

```
// Що саме зберігає ця функція? Незрозуміло з її імені.  
function save() {}
```

Q&A

Види оголошення функцій

Види оголошення функцій

У JavaScript існує кілька способів оголошення функцій:

1. Function Declaration
2. Function Expression
3. Arrow Function (стрілочні функції)
4. IIFE (Immediately Invoked Function Expression)
5. Оголошення функції в об'єкті

Function Declaration

Це найтрадиційніший спосіб оголошення функції. Функція, оголошена таким чином, піднімається, тому її можна викликати до того, як вона була оголошена в коді.

A dark-themed code editor window with three colored window control buttons (red, yellow, green) in the top-left corner. It contains a JavaScript function declaration.

```
function greet() {  
    console.log("Hello, World!");  
}
```

Hoisting

"Hoisting" — це механізм у JavaScript, завдяки якому змінні і оголошення функцій піднімаються до верху їхньої області видимості перед виконанням коду.

```
// Виклик функції перед її оголошенням  
sayHello(); // виводить "Hello!"  
  
function sayHello() {  
    console.log("Hello!");  
}
```


Function Expression

Тут функція оголошується і присвоюється змінній. Ця функція не піднімається, тому її можна викликати лише після оголошення.



```
const greet = function() {  
  console.log("Hello, World!");  
};
```

У чому різниця?

Function Expression створюється, коли виконання доходить до нього, а потім вже може використовуватися.

Function Declaration можна використовувати у всьому скрипті (або блоці коду, якщо функція оголошена у блоці).

Arrow Function

Це більш сучасний спосіб оголошення функцій, який був доданий у ES6. Його основна особливість полягає в коротшому синтаксисі і в особливостях роботи зі словом `this`.

```
const greet = () => {  
  console.log("Hello, World!");  
};  
  
const sum = (a, b) => a + b;  
  
const sum = (a, b) => {  
  return a + b;  
}
```

IIFE

IIFE (Immediately Invoked Function Expression) - ця функція виконується відразу після оголошення.

A dark-themed code editor window with three colored window control buttons (red, yellow, green) in the top-left corner. It contains a single line of JavaScript code that defines an immediately invoked function expression (IIFE).

```
(function () {  
    console.log("Hello world!");  
})();
```

Оголошення функції в об'єкті

Функції можна оголошувати як методи в об'єктах.

```
const obj = {  
  greet1: function() {  
    console.log("Hello, World!");  
  },  
  //  
  greet2() {  
    console.log("Hello, World!");  
  },  
  //  
  greet3: () => {  
    console.log("Hello, World!");  
  }  
};  
  
obj.greet1();
```

Анонімні функції

Анонімні функції — це функції в JavaScript, які не мають імені. Оскільки вони не мають імені, вони не можуть бути викликані за своїм ім'ям після оголошення, але можуть бути використані як callback-функції або можуть бути присвоєні змінним або об'єктам.



```
const greet = function() {  
  console.log("Hello, World!");  
};
```

Чисті функції

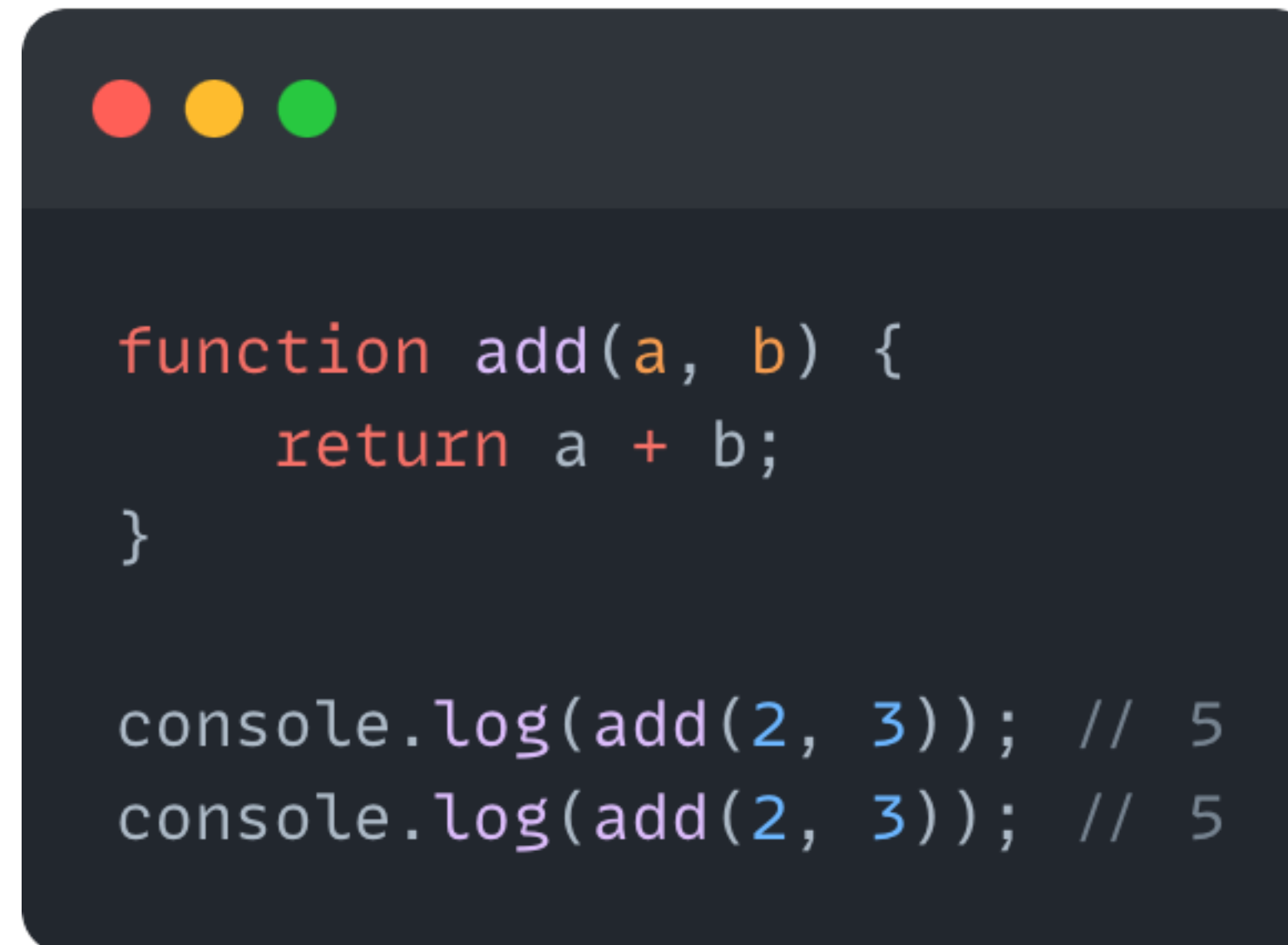
Чисті функції — це поняття в функціональному програмуванні, яке вказує на функції, що відповідають певним критеріям. Ось основні характеристики чистих функцій:

1. **Відсутність побічних ефектів:** Чиста функція не має побічних ефектів. Це означає, що вона не змінює жодних зовнішніх змінних, або не має інших побічних дій (наприклад, не виводить дані на консоль, не робить запити до бази даних тощо).
2. **Віддавання результату на основі вхідних даних:** Вивід функції виключно залежить від її вхідних даних. Для одних і тих же вхідних значень чиста функція завжди повертає одне й те ж результат.
3. **Відсутність глобального стану:** Чиста функція не залежить від і не змінює зовнішній (глобальний) стан.

Чисті функції

Ось приклад чистої функції.

Ця функція завжди повертає однаковий результат для одних і тих же значень *a* та *b* і не має побічних ефектів.

A dark-themed code editor window with three colored window control buttons (red, yellow, green) in the top-left corner. It contains JavaScript code for a pure function.

```
function add(a, b) {  
    return a + b;  
}  
  
console.log(add(2, 3)); // 5  
console.log(add(2, 3)); // 5
```


Q&A

Рекурсія

Рекурсія

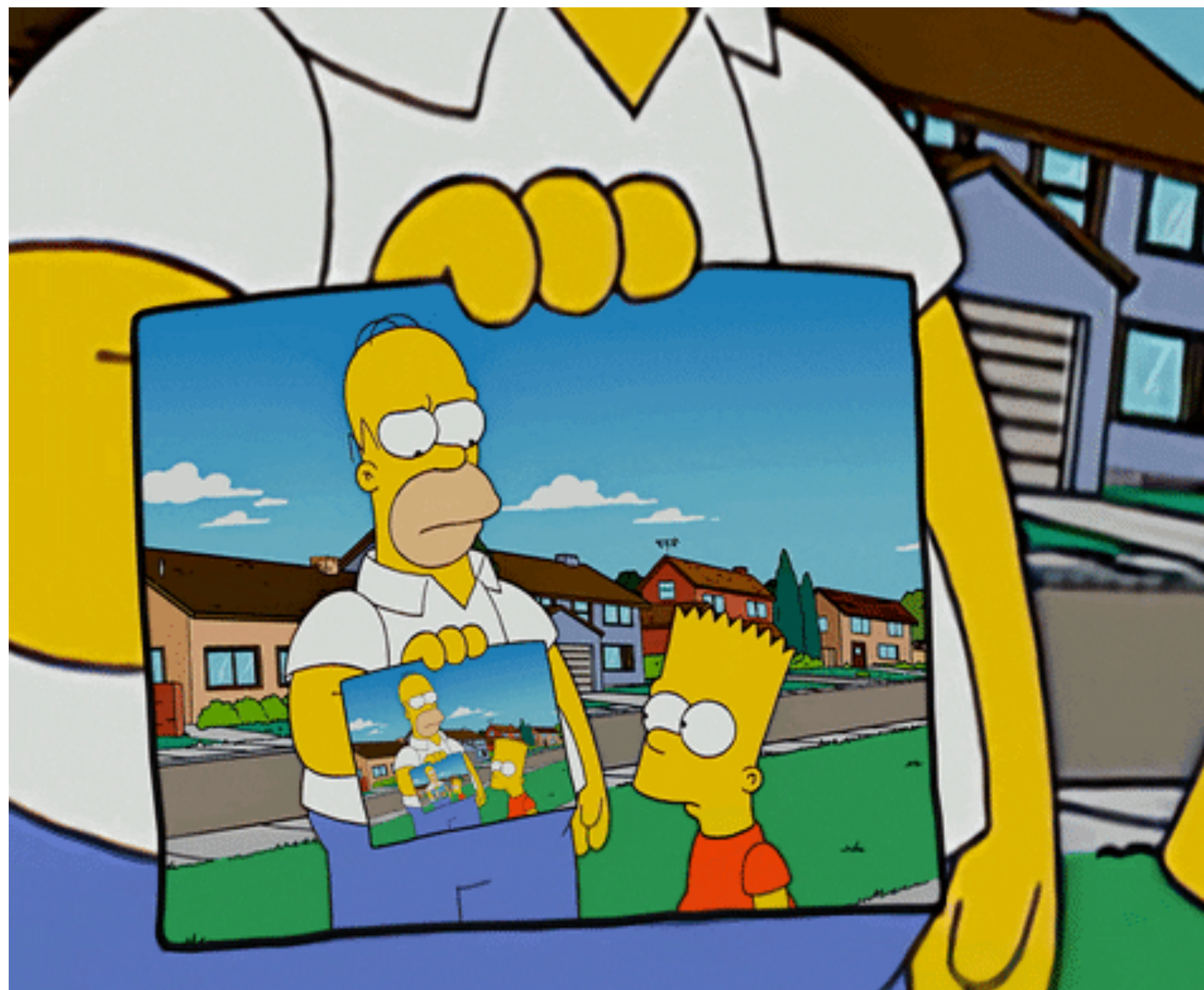
Рекурсія - це прийом у програмуванні, корисний у ситуаціях, коли завдання може бути розділена на кілька аналогічних, але більш простих завдань.

Рекурсія в програмуванні відноситься до техніки, за допомогою якої функція викликає сама себе, прямо або опосередковано, для вирішення задачі чи підзадачі.

Рекурсія часто використовується для розбиття великих чи складних задач на менші підзадачі того ж самого типу. Ключова ідея рекурсії полягає в тому, що простіші випадки задачі можуть бути вирішені безпосередньо, тоді як більш складні випадки можуть бути розбиті на менші підзадачі.

Приклади з життя

CREATIVE
& TECH
PRJCTR
ONLINE
INSTITUTE



Рекурсія

Коли використовуєте рекурсію, важливо визначити один або декілька **базових випадків**, які визначають умови припинення рекурсивних викликів.

Якщо базовий випадок не визначено або не досягається, рекурсивна функція може викликатися нескінченно, призводячи до переповнення стека.

Рекурсія - приклад

Давайте розглянемо функцію `pow(x, n)`, яка використовує ітеративний підхід до вирішення задачі (тобто цикли).

```
function pow(x, n) {  
  let result = 1;  
  
  // множимо result на x n разів у циклі  
  for (let i = 0; i < n; i++) {  
    result *= x;  
  }  
  
  return result;  
}
```

Рекурсія - приклад

Розглянемо приклад рекурсивного рішення:

```
function pow(x, n) {  
  if (n === 1) {  
    return x;  
  }  
  
  return x * pow(x, n - 1);  
}
```

Рекурсія - факторіал

Для ілюстрації рекурсії розглянемо приклад рекурсивного обчислення факторіалу числа:

```
function factorial(n) {  
  if (n === 0) {  
    return 1; // базовий випадок  
  }  
  return n * factorial(n - 1); // рекурсивний виклик  
}
```


Рекурсія - факторіал

Ось як ця функція працює:

`factorial(0)` повертає `1` (базовий випадок).

`factorial(1)` повертає `1 * factorial(0)` (рекурсивний випадок), що дорівнює `1`.

`factorial(2)` повертає `2 * factorial(1)`, що дорівнює `2`.

`factorial(3)` повертає `3 * factorial(2)`, що дорівнює `6`.

Рекурсія - висновки

Будь-яка рекурсія може бути перероблена з використанням ітеративного підходу (циклів).

Попри свою потужність, рекурсія може бути менш ефективною за ітеративні підходи до вирішення тих же задач у деяких ситуаціях, тому важливо обирати рекурсію обережно і розуміти її потенційні витрати.

Рекурсія має обмеження по глибині (10000 викликів).

Q&A

Summary

01 / Дізнались що таке функції та яка їх структура

02 / Дізнались про види функцій і їх відмінності

03 / Познайомились з терміном "рекурсія" та написали приклади її використання



What's next?

CREATIVE
& TECH
PRJCTR
ONLINE
INSTITUTE

01 / Просунуті методи масивів

02 / Об'єкт Date

prjctr.com

Додаткові матеріали

01 / [Про function declaration та function expression](#)

02 / [Про рекурсію](#)

Q&A