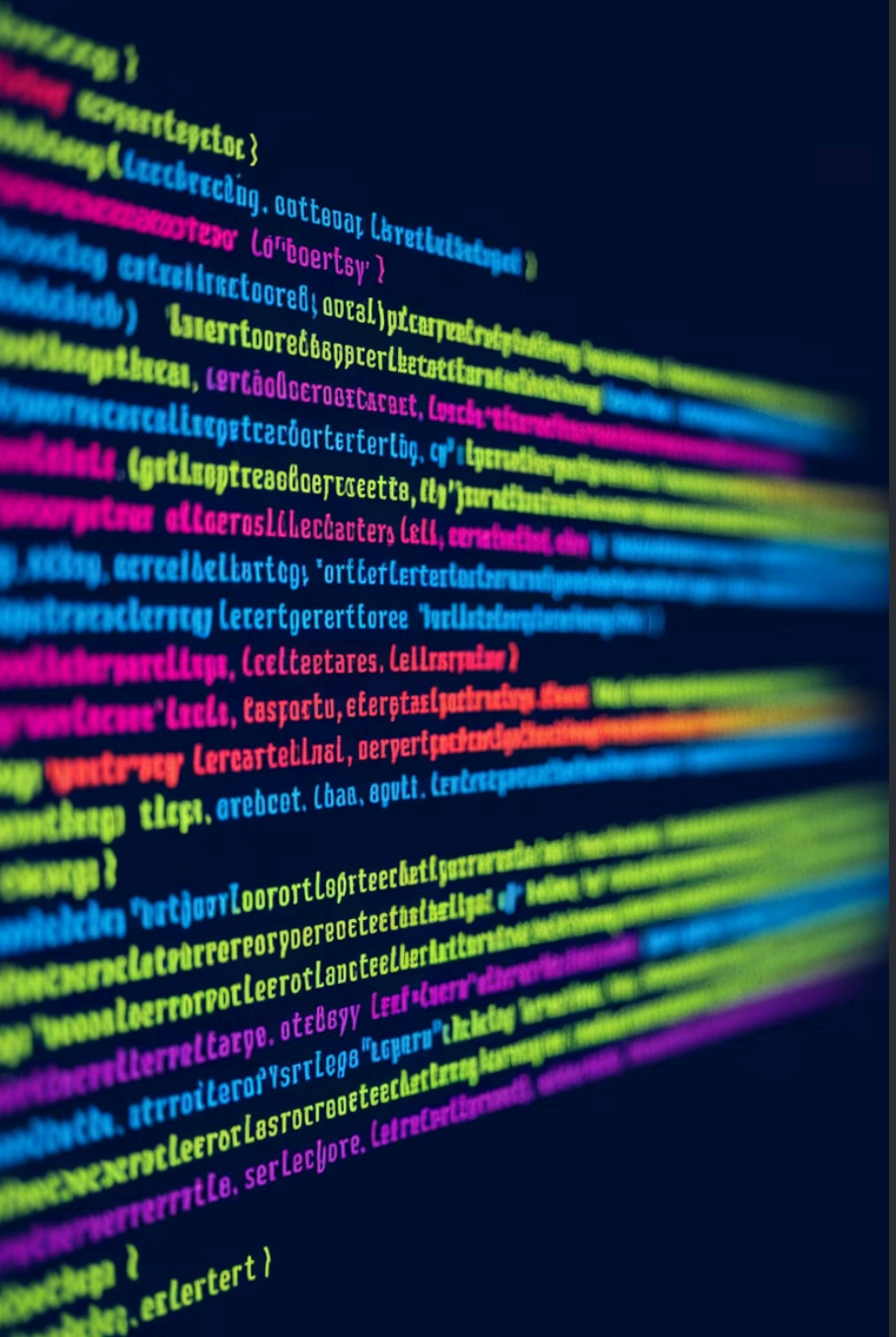


Масиви та Цикли в JavaScript

Ласкаво просимо на четверту лекцію курсу вивчення JavaScript! У цій презентації ми розглянемо масиви та цикли, включаючи вкладені масиви, доступ до даних масиву, зміну даних масиву, доступ до багатовимірних масивів та різні типи циклів.

Ці основні концепції допоможуть вам розпочати програмування на JavaScript та підготують вас до більш складних тем у майбутніх лекціях.

by o d

Вкладені масиви

Що таке вкладені масиви?

Масиви можуть містити інші масиви як свої елементи, утворюючи багатовимірні структури. Це зручно для представлення таблиць, матриць або складних даних.

Приклад вкладеного масиву

```
let nestedArray = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];

console.log(nestedArray[0][1]); // Виведе: 2
```

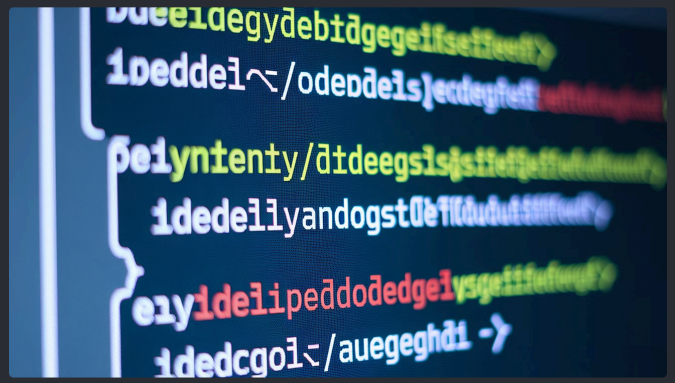
Практичне застосування

Вкладені масиви використовуються для зберігання даних у табличному форматі, створення матриць для математичних обчислень та моделювання складних структур даних.

Доступ до даних масиву

Індексація з нуля

Елементи масиву доступні за індексами, починаючи **з нуля.**



Синтаксис доступу

Використовуйте квадратні дужки з індексом: array[2]. Приклад:

```
let array = [10, 20, 30];
console.log(array[1]); // 20
```

Довжина масиву

Властивість **length** повертає кількість елементів у масиві, що дозволяє ітерувати через всі елементи.

Зміна даних масиву

1

Присвоєння за індексом

Елементи масиву змінюються шляхом присвоєння нового значення: array[1] = 50. Це оновлює дані без створення нового масиву.

2

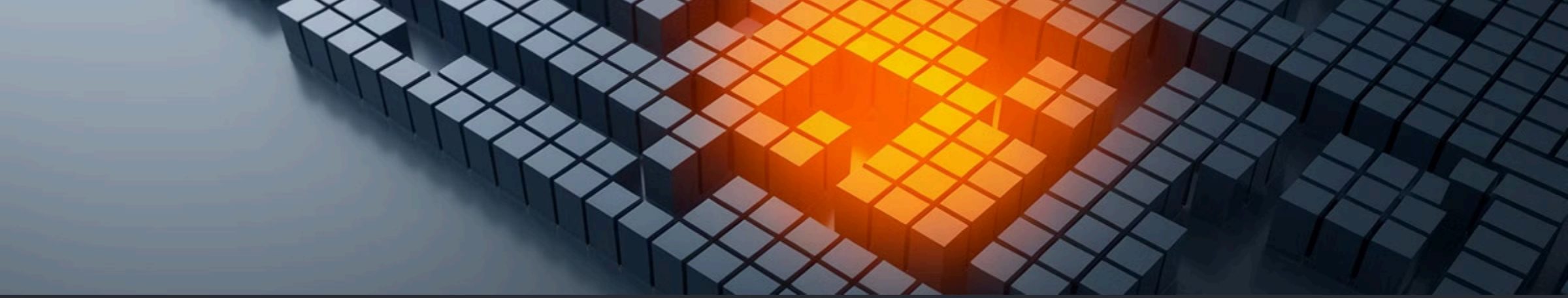
Приклад зміни

array[2] = 35; // Змінює третій елемент на 35

3

Динамічне розширення

Масиви JavaScript розширюються автоматично при присвоєнні значення за індексом поза поточними межами.



Доступ до багатовимірних масивів

1

Перший рівень індексації

Перший індекс вказує на зовнішній масив, визначаючи, який вкладений масив буде використовуватися.

2

Другий рівень індексації

Другий індекс вказує на елемент у вибраному вкладеному масиві, дозволяючи отримати конкретне значення.

3

Приклад доступу

```
let multiDimensionalArray = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
console.log(multiDimensionalArray[1][2]); // Виведе: 6
```

Основні методи масивів: push() і pop()

push()

Метод push() додає один або кілька елементів до кінця масиву і повертає нову довжину масиву.

let fruits = ["яблуко", "банан"]; fruits.push("апельсин"); console.log(fruits); // ["яблуко", "банан", "апельсин"]

pop()

Метод pop() видаляє останній елемент з масиву і повертає його.

let fruits = ["яблуко", "банан", "апельсин"]; let lastFruit = fruits.pop(); console.log(lastFruit); // "апельсин" console.log(fruits); // ["яблуко", "банан"]

Основні методи масивів: shift() і unshift()

shift()

Метод shift() видаляє перший елемент масиву, а решта елементів зсуваються на одну позицію вліво.

let fruits = ["яблуко", "банан", "апельсин"]; let firstFruit = fruits.shift(); console.log(firstFruit); // "яблуко" console.log(fruits); // ["банан", "апельсин"]

unshift()

Метод unshift() додає один або кілька елементів на початок масиву, зсуваючи всі інші елементи вправо.

let fruits = ["банан", "апельсин"]; fruits.unshift("яблуко"); console.log(fruits); // ["яблуко", "банан", "апельсин"]

Ітераційні методи масивів: map()

Метод map() створює новий масив з результатами виклику заданої функції для кожного елемента масиву, не змінюючи оригінальний масив.

Трансформація елементів

map() застосовує функцію до кожного елемента масиву та створює новий масив з результатів, зберігаючи оригінальні дані незмінними.

Функція зворотного виклику

Функція, яка передається методу map(), викликається для кожного елемента масиву з трьома аргументами: поточний елемент, індекс та сам масив.

```
let numbers = [1, 2, 3, 4];
let doubled = numbers.map(function(num) {
    return num * 2;
});
console.log(doubled); // [2, 4, 6, 8]
```

Ітераційні методи масивів: filter()

Метод filter() створює новий масив з усіма елементами, які пройшли перевірку, задану у функції.

1

Фільтрація елементів

Метод filter() перевіряє кожен елемент масиву і залишає тільки ті, що відповідають умові.

2

Функція-предикат

Функція, яка передається методу filter(), повинна повертати булеве значення, що визначає, чи включати елемент у новий масив.

3

Новий масив

Як результат filter() повертає новий масив, не змінюючи оригінальний.

```
let numbers = [1, 2, 3, 4];
let evenNumbers = numbers.filter(function(num) {
    return num % 2 === 0;
});
console.log(evenNumbers); // [2, 4]
```

Ітераційні методи масивів: reduce()

Метод reduce() застосовує функцію до кожного елемента масиву (зліва направо), щоб звести його до одного значення.

1 2 3

Принцип роботи

Reduce обробляє масив послідовно, зберігаючи проміжний результат в акумуляторі та передаючи його далі.

Акумулятор

Перший параметр функції зворотного виклику — акумулятор, який зберігає результат попередніх обчислень.

Поточний елемент

Другий параметр — поточний елемент масиву, який обробляється на даному кроці ітерації.

```
let numbers = [1, 2, 3, 4];
let sum = numbers.reduce(function(total, num) {
    return total + num;
}, 0);
console.log(sum); // 10
```

У цьому прикладі ми підсумовуємо всі елементи масиву, починаючи з початкового значення 0.

Цикл while

Цикл while є основною конструкцією для повторення коду в JavaScript, що працює за певним принципом:

1

Умова виконання

Виконує код доти, доки задана умова залишається істинною.

2

Перевірка умови

Умова перевіряється перед кожною ітерацією циклу.

3

Приклад використання

let i = 0; while (i < 5) { console.log("Цикл while: " + i); i++; }

При роботі з циклом while важливо забезпечити умову завершення, щоб уникнути нескінченного виконання.

Цикл do...while

1

Гарантоване виконання

Виконує код хоча б один раз

2

Виконання блоку коду

Код всередині блоку виконується

3

Перевірка умови

Умова перевіряється після виконання

На відміну від циклу while, цикл do...while завжди виконує свій блок коду принаймні один раз, незалежно від умови, і перевіряє її лише після виконання першої ітерації. Це корисно, коли потрібно гарантувати, що певний код буде виконано хоча б один раз.

```
let j = 0;
do {
  console.log(j);
  j++;
} while (j < 5);
```

Цикл for

1

Ініціалізація

Встановлення початкового значення лічильника

2

Умова

Перевірка умови продовження циклу

3

Оновлення

Зміна лічильника після кожної ітерації

Цикл for є найбільш компактним способом створення циклу з визначеною кількістю ітерацій. Він включає ініціалізацію, перевірку умови і оновлення змінних у кожній ітерації, все в одному рядку.

Приклад: for (let k = 0; k < 5; k++) { console.log("Цикл for: " + k); }

Цикли for...of та for...in

Цикл for...of

Використовується для ітерації по елементах масивів або інших ітеративних об'єктів, забезпечуючи зручний доступ до значень.

```
let iterableArray = [10, 20, 30, 40, 50];

for (let value of iterableArray)

{ console.log("Цикл for...of: " + value);

}
```

Цикл for...in

Ітерація по ключах об'єкта, що дозволяє працювати з усіма його властивостями.

```
let object = {a: 1, b: 2, c: 3};

for (let key in object)

{ console.log("Цикл for...in: " + key + " = " + object[key]);

 }
```

Підсумок: Масиви та Цикли в JavaScript

У цій лекції ми розглянули основні концепції роботи з масивами та циклами в JavaScript, які є фундаментальними для програмування.

Структура масивів

Вкладені та багатовимірні масиви, доступ та зміна даних

Основні методи

Push(), pop(), shift(), unshift() для модифікації масивів

Типи циклів

While, do...while, for, for...of, for...in для контролю потоку

Ітераційні методи

Map(), filter(), reduce() для обробки даних масиву

Ці знання є взаємопов'язаними та доповнюють одне одного, створюючи потужний інструментарій для ефективної роботи з даними. Вони будуть використовуватися як основа для розуміння більш складних концепцій у майбутніх лекціях.