Java Script

NEXT

GENIUS space

План

Масиви

методи масиву

Цикли

break и continue

області видимості

Масиви

Масив - структура даних для зберігання та маніпулювання колекцією індексованих значень. Використовуються для зберігання впорядкованих колекцій даних, наприклад, списку курортів, товарів, клієнтів у готелі тощо.

Синтаксис для створення нового масиву - квадратні дужки [] зі списком елементів розділених комами. У масиві може зберігатися будь-яка кількість елементів будь-якого типу.

Оголошення

Існує два типи синтаксу для створення порожнього масиву:

```
let arr = new Array();
let arr = [];
```

Майже завжди використовують другий тип синтаксису. Ми можемо вказати початкові елементи масиву у квадратних дужках:

```
let fruits = ["Apple", "Orange", "Plum"];
```

Елементи масиву нумеруються починаючи з нуля.

Ми можемо отримати елемент масиву, вказавши його номер в квадратних дужках:

```
let fruits = ["Apple", "Orange", "Plum"];

alert( fruits[0] ); // Apple
alert( fruits[1] ); // Orange
alert( fruits[2] ); // Plum
```

Перевизначення
Елементи масиву можна замінювати та додавати, звертаючись до елемента масиву за індексом.

```
fruits[2] = 'Pear';
```

...Або додати новий:

```
fruits[3] = 'Lemon'; // тепер ["Apple", "Orange", "Pear", "Lemon"]
```

Загальна кількість елементів масиву зберігається у його властивості length:

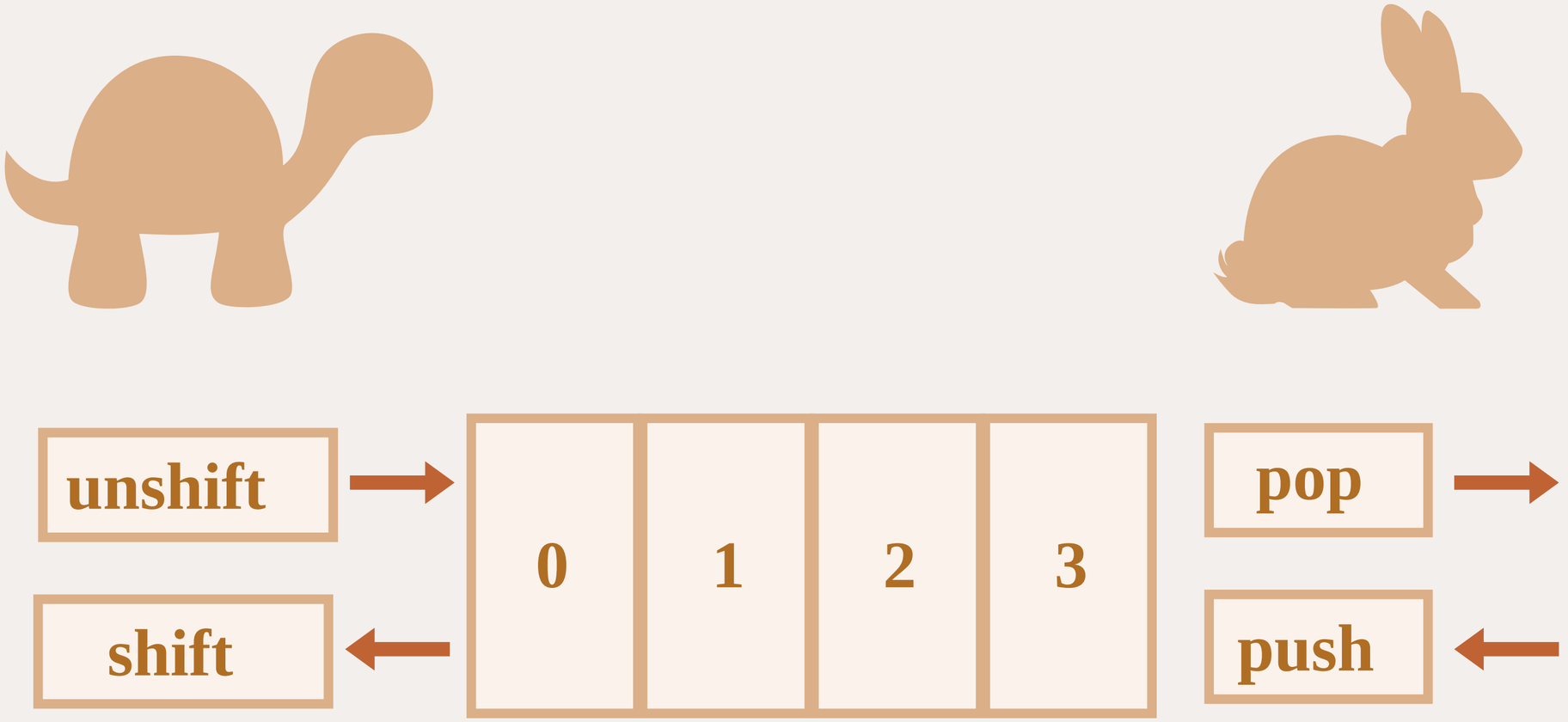
```
let fruits = ["Apple", "Orange", "Plum"];
alert( fruits.length ); // 3
```

У масивах можуть зберігатись елементи будь-якого типу.

```
// різні типи значень
let arr = [ 'Apple', { name: 'John' }, true, function() { alert('hello'); } ];
```

Методи масиву

Методи push(), pop(), shift(), unshift()



push() — дозволяє додати один або кілька елементів до кінця масиву. Метод повертає значення властивості length, що визначає кількість елементів у масиві.

pop() — видаляє елемент із кінця масиву

shift() — видаляє елемент із початку масиву та повертає його (елемента) значення

unshift() — додає елемент на початок масиву.

splice - Метод arr.splice – це універсальний «швейцарський ніж» для роботи з масивами. Вміє все: додавати, видаляти і замінювати елементи.

синтаксис: arr.splice(position, num)

```
let arr = ["I", "study", "JavaScript"];
arr.splice(1, 1); // з індексу 1 видалимо 1 елемент
```

position — вказує позицію (індекс) першого елемента видалення
num - визначає кількість елементів, що видаляються
Метод splice змінює вихідний масив та повертає масив, що містить віддалені елементи.

Метод slice() - Синтаксис методу slice() однаковий для рядків та масивів. Його просто запам'ятати. Він дозволяє витягувати елементи підмножини масиву і додавати в новий масив. Найчастіше використовується для створення копії частини або цілого вихідного масиву.

arr.slice(begin, end. - Копіює елементи від begin до, але не включаючи, end.

Обидва аргументи (begin та end) не обов'язкові.

Параметр begin визначає індекс, з якого слід розпочинати вилучення.

Параметр end визначає індекс елемента, на якому слід закінчити вилучення.

Метод slice не включає елемент з ідексом end у вилучені елементи.

Якщо begin і end не вказані, копіювання буде з початку масиву, з індексу 0, тобто скопіюється весь масив.

Якщо не вказати параметр end, метод slice використовуватиме довжину масиву (length) для параметра end.

Метод slice не змінює вихідний масив, а повертає новий масив, що містить копію елементів вихідного.

Можна використовувати негативні індекси, вони відраховуються з кінця.

concat

Метод arr.concat створює новий масив, в який копіює дані з інших масивів та додаткові значення.

arr.concat(arg1, arg2...)
Він приймає будь-яку кількість аргументів – масивів або значень.

Результатом є новий масив, що містить елементи з arr, потімarg1, arg2 тощо. Якщо аргумент argN є масивом, то всі його елементи копіюються. В іншому випадку буде скопійовано сам аргумент.

```
const oldClients = ['Mango', 'Ajax', 'Poly', 'Kiwi'];
const newClients = ['Monkong', 'Singu'];

const allClients = oldClients.concat(newClients);

console.log(allClients);
// ["Mango", "Ajax", "Poly", "Kiwi", "Monkong", "Singu"]
```

Методи arr.indexOf, arr.lastIndexOf та arr.includes мають однаковий синтаксис і роблять по суті те ж саме, що і їх рядкові аналоги, але працюють з елементами замість символів:

arr.indexOf(item, from) – шукає item, починаючи з індексу from, і повертає індекс, на якому був знайдений шуканий елемент, в іншому випадку‑1.

arr.lastIndexOf(item, from) – те ж саме, але шукає справа наліво.

arr.includes(item, from) – шукає item, починаючи з індексу from, і повертає true, якщо пошук успішний.

Часте завдання програмування – виконання однотипної дії багато разів. Наприклад, вивести клієнтів зі списку один за одним, або перебрати суми зарплат і для кожної виконати однаковий код. Саме для таких цілей багаторазового повторення однієї ділянки коду використовуються цикли.

Цикл — конструкція, що управляє, у високорівневих мовах програмування, призначена для організації багаторазового виконання набору інструкцій.

Тіло циклу – послідовність інструкцій, призначена для багаторазового виконання.

Ітерація – одиничне виконання тіла циклу.

Умова виходу - вираз, що визначає в черговий раз виконуватися ітерація, або цикл завершиться.

Лічильник - змінна, що зберігає поточний номер ітерації. Цикл необов'язково містить лічильник, і не повинен бути один, умова виходу з циклу може залежати від кількох змінних у циклі змінних.

Виконання будь-якого циклу включає:
початкову ініціалізацію змінних циклу
перевірку умови виходу
виконання тіла циклу
оновлення змінної циклу на кожній ітерації

Цикл “while”

Цикл while має такий синтаксис:

```
while (умова) {
  // код
  // так зване "тіло циклу"
}
```

Доки умова є вірною, виконується код із тіла циклу.

Наприклад, цикл нижче виводить і поки і < 3:

```
let i = 0;
while (i < 3) { // показується 0, далі 1, потім 2
  alert( i );
  i++;
}
```

Цикл “do...while”

Перевірка умови може бути переміщена нижче тіла циклу використовуючи do..while синтаксис:

```
do {
  // тіло циклу
} while (умова);
```

Цикл спочатку виконує тіло, а потім перевіряє умову, і поки умова є true, цикл виконується знову і знову.

Наприклад:

```
let i = 0;
do {
  alert( i );
  i++;
} while (i < 3);
```

Цикл for

Цикл із лічильником — цикл, у якому деяка змінна змінює своє значення від заданого початкового до кінцевого значення, з деяким кроком, і кожного значення цієї змінної тіло циклу виконується один раз.

У більшості процедурних мов програмування реалізується оператором for, у якому вказується лічильник, необхідну кількість ітерацій та крок, з яким змінюється лічильник.

Алгоритм виконання циклу for:

Ініціалізація (initialization) - вираз ініціалізації виконується один раз, коли починається цикл. Використовується для ініціалізації змінної-лічильника. Якщо використовується ключове слово let, змінна лічильника є локальною для циклу.

Умова (condition, test) - вираз, що оцінюється перед кожною ітерацією циклу. Тіло циклу виконується лише тоді, коли вираз умови набуває значення true. Цикл завершується, якщо значення буде false.

Тіло (statements) – виконується у разі задоволення умови.

Пост-вираз (post-expression) - виконується після тіла кожної ітерації циклу, але перед перевіркою умови. Використовується для оновлення змінної-лічильника.

```
const max = 10;

for (let i = 0; i < max; i += 1) {
  console.log(i);
}
```

Переривання циклу: “break”
Зазвичай, цикл завершується, коли умова стає false.

Але ми можемо в будь-який момент вийти з циклу, використавши спеціальну директиву break.

```
for (let i = 0; i < 10; i += 1) {
    if (i === 5) {
  console.log('5 ітерація!');
      break;
    }
}
```

Продовження з наступної ітерації
Директива continue — це “полегшена версія” break. Вона не зупиняє весь цикл. Натомість, вона зупиняє поточну ітерацію і починає виконання циклу спочатку з наступної ітерації (якщо умова циклу досі вірна).

Її зручно використовувати коли закінчили з поточною ітерацією і хочемо продовжити з наступної.

Цикл нижче використовує continue щоб вивести лише непарні значення:

```
for (let i = 0; i < 10; i++) {

  // якщо умова справджується, тоді пропускаємо решту тіла циклу і починаємо з наступної ітерації
  if (i % 2 == 0) continue;

  alert(i); // 1, потім 3, 5, 7, 9
}
```

Область видимості
Область видимості змінних (variable scope) – доступність змінних у певному місці коду. Є кілька областей видимості: глобальна, блокова, eval та функції.

Глобальна область видимості використовується за умовчанням. Усі і всі мають доступ до змінних оголошених у ній. Змінні оголошені у глобальній області видимості вразливі, оскільки може змінити будь-яку ділянку коду.

Будь-яка конструкція використовує фігурні дужки {} (умови, цикли, функції тощо) створює нову локальну область видимості, і змінні, оголошені в цій області видимості, використовуючи let або const, не доступні поза цим блоком.


Глобальная область видимости
Блочная область видимости A
Блочная область видимости B
Блочная область видимости C

Домашнє завдання

```
//  -- 1 --
// У вас є масив об’єктів fruts, і в кожному з них є name
// Напишіть код, який перетворює їх в масив імен.

// const fruts = [
//   { id: 0, name: "Apple" },
//   { id: 1, name: "Tomat" },
//   { id: 2, name: "Cherry" },
//   { id: 3, name: "Orange" },
// ];

//  -- 2 --
//Виведіть парні числа від 2 до 10, використовуючи цикл for.

//  -- 3 --
//Замініть цикл "for" на "while"
// for (let i = 0; i < 5; i++) {
//     console.lpg( `цифра ${i}!` );
//   }
```

Домашнє завдання

```
//  -- 4 --
//Напишіть цикл, який пропонує prompt ввести число більше за 100.
//Якщо відвідувач введе менше число – попросити ввести ще раз, і так далі.
//Цикл повинен запитувати число доти, доки відвідувач не введе число,
// більше за 100, або не скасує ввід/введе порожній рядок.

//  -- 5 --
// Вирахуйте середній вік

// const girls = [
//   { age: 23, name: "Оля" },
//   { age: 29, name: "Аня" },
//   { age: 10, name: "Юля" },
//   { age: 20, name: "Катя" },
// ];
```

Додаткові матеріали

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array?retiredLocale=uk

(дуже раджу випробувати всі методи)

- https://medium.com/@mandeepkaur1/a-list-of-javascript-array-methods-145d09dd19a0
- https://www.w3schools.com/jsref/jsref_obj_array.asp
- https://uk.javascript.info/array
- https://uk.javascript.info/array-methods
- https://uk.javascript.info/iterable
- https://uk.javascript.info/while-for
- http://xn--80adth0aefm3i.xn--j1amh/%D1%86%D0%B8%D0%BA%D0%BB%D0%B8