

Повторим;)

■ Как получить доступ к значению свойства объекта?

■ Как удалить свойство из объекта?

■ Как проверить есть ли некоторое свойство в объекте?

■ Когда требуется применение
брекет-синтаксиса?

■ Какие типы относятся к
ссылочным (reference type)?

JS: Array Methods

НАШИ ПРАВИЛА




Включенная камера




Вопросы по поднятой руке



Не перебиваем друг друга



Все вопросы, не связанные с тематикой курса (орг-вопросы и т. д.), должны быть направлены куратору



Подготовьте свое рабочее окружение для возможной демонстрации экрана (закройте лишние соцсети и прочие приложения)

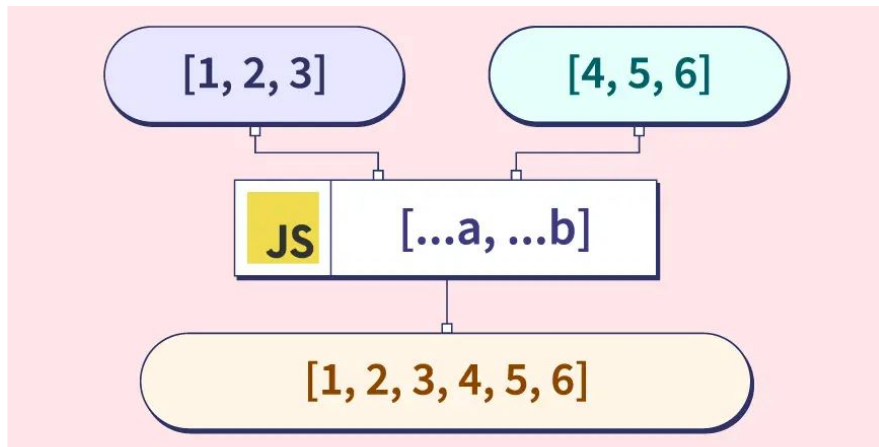
ЦЕЛЬ

Изучить методы массивов

ПЛАН ЗАНЯТИЯ

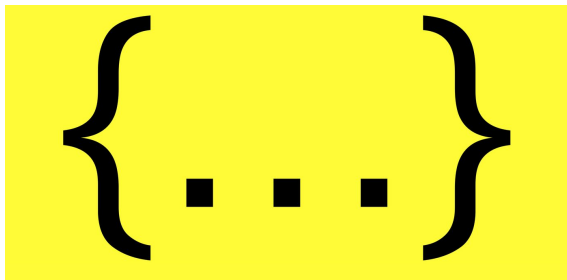
- Деструктуризация
- Методы массивов

Деструктуризация



Оператор расширения - spread

В JavaScript оператор расширения (spread operator) представляет собой синтаксическую конструкцию, которая используется для создания копий массивов, объединения массивов, передачи аргументов функции и других подобных задач.



1. Для массивов:

В приведенном примере `...arr1` разворачивает элементы массива `arr1` в новом массиве `arr2`.

```
const arr1 = [1, 2, 3];  
const arr2 = [...arr1, 4, 5, 6];  
console.log(arr2); // [1, 2, 3, 4, 5, 6]
```


2. Для объектов:

Здесь spread оператор используется для создания нового объекта obj2, который содержит все свойства из obj1, а также новые свойства.

```
const obj1 = { key1: 'value1', key2: 'value2' };  
const obj2 = { ...obj1, key3: 'value3', key4: 'value4' };  
console.log(obj2);  
// { key1: 'value1', key2: 'value2', key3: 'value3', key4: 'value4' }
```

3. Передача аргументов функции

В данном примере spread оператор используется для передачи элементов массива в качестве аргументов функции `sum`.

```
function sum(a, b, c) {  
  return a + b + c;  
}  
  
const numbers = [1, 2, 3];  
console.log(sum(...numbers)); // 6
```

Деструктурирующее присваивание

A yellow rectangular box containing the text `[{ ... }]` in black font, representing the syntax for destructuring an object from an array.

Деструктурирующее присваивание – это специальный синтаксис, который позволяет нам «распаковать» массивы или объекты в несколько переменных, так как иногда они более удобны.



Деструктуризация массива

```
1 // у нас есть массив с именем и фамилией
2 let arr = ["Ilya", "Kantor"];
3
4 // деструктурирующее присваивание
5 // записывает firstName = arr[0]
6 // и surname = arr[1]
7 let [firstName, surname] = arr;
8
9 alert(firstName); // Ilya
10 alert(surname);  // Kantor
```

Деструктурирующее присваивание ничего не уничтожает, его задача – только скопировать нужные значения в переменные.

Деструктуризация объекта

Деструктурирующее присваивание также работает с объектами.

```
1  let options = {  
2    title: "Menu",  
3    width: 100,  
4    height: 200  
5  };  
6  
7  let {title, width, height} = options;  
8  
9  alert(title); // Menu  
10 alert(width); // 100  
11 alert(height); // 200
```

Для явного задания имени переменной используем синтаксис:
название ключа: имя переменной

```
let {title: optionName, width, height} = options;
```

Map

Метод `map()` создает новый массив с результатами вызова предоставленной функции для каждого элемента в массиве.

```
const numbers = [1, 2, 3, 4, 5];  
const squaredNumbers = numbers.map(num => num ** 2);  
// squaredNumbers: [1, 4, 9, 16, 25]
```

ForEach

Метод `forEach()` выполняет предоставленную функцию один раз для каждого элемента массива.

Возвращаемое значение:
undefined

```
const numbers = [1, 2, 3, 4, 5];  
numbers.forEach(num => console.log(num));  
  
// Output: 1, 2, 3, 4, 5
```


Filter

Метод `filter()` создает новый массив с элементами, для которых предоставленная функция возвращает `true`

```
const students = [
  { name: "Alice", grade: 85 },
  { name: "Bob", grade: 90 },
  { name: "Charlie", grade: 78 },
];

const highGrades = students.filter((student) => student.grade
> 80);

// highGrades: [{ name: 'Alice', grade: 85 }, { name: 'Bob',
grade: 90 }]
```

reverse

Описание:

Метод `reverse()` на месте обращает порядок следования элементов массива. Первый элемент массива становится последним, а последний — первым.

Синтаксис:

```
array.reverse()
```

```
var myArray = ["один", "два", "три"];
```

```
myArray.reverse();
```

```
console.log(myArray); // ['три', 'два', 'один']
```

join

Описание:

Метод `join()` преобразует все элементы массива в строки и объединяет их в одну большую строку.

Синтаксис:

`arr.join(separator)`

```
1 let arr = ['Вася', 'Петя', 'Маша'];
2
3 let str = arr.join(';'); // объединить массив в строку через ;
4
5 alert( str ); // Вася;Петя;Маша
```

Reduce

Метод `reduce()` применяет функцию к аккумулятору и каждому значению массива (слева направо), сводя массив к одному значению.

```
const numbers = [1, 2, 3, 4, 5];  
const sum = numbers.reduce((acc, num) => acc + num, 0);  
// sum: 15
```

Reduce

```
[0, 1, 2, 3, 4].reduce(function (previousValue, currentValue, index, array) {  
    return previousValue + currentValue;  
});
```

	previousValue	currentValue	index	array	возвращаемое значение
первый вызов	0	1	1	[0, 1, 2, 3, 4]	1
второй вызов	1	2	2	[0, 1, 2, 3, 4]	3
третий вызов	3	3	3	[0, 1, 2, 3, 4]	6
четвёртый вызов	6	4	4	[0, 1, 2, 3, 4]	10

Значение, возвращённое методом `reduce()` будет равным последнему результату выполнения колбэк-функции — **10**.



Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH