Front-end

JavaScript

JS. Arrays

- Перебирающие методы массивов

JS. Arrays

- copyWithin()
- every()
- some()
- find()
- findIndex()
- sort()
- forEach()
- map()
- filter()
- reduce()

JS. Arrays

copyWithin()

Позволяет скопировать элементы массива (в зависимости от их индекса) и вставить их в тот же массив, заменяя определенные элементы массива (в зависимости от их индекса), длина массива при этом не изменяется

Синтаксис

arr.copyWithin(target, start, end)

Возвращаемое значение

Измененный массив.

JS. Arrays

copyWithin()

Параметры

target - обязательный. Позиция индекса, в который будет вставлен копируемый элемент

start - необязательный. Позиция индекса для начала копирования элементов (по умолчанию 0)

end - необязательный. Позиция последнего индекса для копирования элементов (по умолчанию — array.length)

```
const colors = ["Green", "Red", "Blue", "Pink"];

colors.copyWithin(2,0); // => Green,Red,Green,Red
```

JS. Arrays

every()

Метод массива .every() позволяет узнать, удовлетворяют ли все элементы в массиве условию в функции-колбэке. Результатом вызова метода .every() будет boolean-значение true или false. Если хотя бы один элемент не будет удовлетворять условию, то результат будет false.

Метод every() выполняет функцию один раз для каждого элемента, присутствующего в массиве:

Если он находит элемент массива, где функция возвращает значение false, то every() возвращает false (и не проверяет остальные значения)

Если false не встречается, то функция every() возвращает true

Примечание: every() не выполняет функцию для элементов массива без значений.

Примечание: every() не изменяет исходный массив

JS. Arrays

every()

Синтаксис

```
array.every(function(currentValue, index, arr), thisValue)
```

```
const users = [
 { name: 'Анна', online: true },
 { name: 'Михаил', online: true },
 { name: 'Саша', online: true },
]
const isAllUsersOnline = users.every(user => {
 return user.online
})
console.log(isAllUsersOnline) // => true
```

JS. Arrays

every()

Из-за того, что результат выполнения метода Array.every – это boolean-значение, метод можно удобно использовать прямо в условных конструкциях:

```
const drinks = ['🍺', '🍺', '🍺', '🍺', '🍺']

if (drinks.every(drink => drink === '🍺')) {
  console.log('This is a beer party! 🌮')
}
```

JS. Arrays

some()

Метод массива some() позволяет узнать, есть ли в массиве хотя бы один элемент, удовлетворяющий условию в функции-колбэке. Колбэк-функция будет вызываться для каждого элемента массива до тех пор, пока не вернётся true, либо пока не закончатся элементы массива.

Результатом вызова метода some() будет boolean-значение true или false. Если ни один элемент в массиве не удовлетворит условию, то результат будет false.

JS. Arrays

some()

Синтаксис

const balls = ['🎾', '🏈', '🎾', '🎾']

const areAllBallsGreen = balls.some((ball, index, arr) => ball === '🏈')

console.log(areAllBallsGreen) // => true

В параметра. Если эти параметры есть (они нфункцию можно передавать три е обязательны), то в первый автоматически попадет элемент массива, во второй попадет его номер в массиве (индекс), а в третий - сам массив.

JS. Arrays

some()

Метод .some() перебирает исходный массив и возвращает true, если хотя бы один из элементов массива удовлетворяет нашему условию.

Проверим, содержит ли массив, хотя бы одно число больше 20:

```
const numbers = [20, 16, 11, 13, 15];
const overTwenty = numbers.some((number) => {
  return number > 20;
});
console.log(overTwenty); // => false
```

JS. Arrays

find()

Метод массива find() вернёт первый найденный в массиве элемент, который подходит под условие в переданной колбэк-функции. Если в массиве не найдётся ни одного подходящего элемента, то вернётся значение undefined.

Функция вызывается один раз для каждого элемента, присутствующего в массиве, до тех пор, пока она не вернёт логическое значение true, при этом значение этого элемента возвращается немедленно (не проверяет оставшиеся значения), или до тех пор пока она не достигнет конца массива, возвращая при этом undefined.

Метод find() не изменяет массив для которого он был вызван.

JS. Arrays

find()

```
const numbers = [1, 2, 3, 4, 5, 6];


const result = numbers.find(function isElementEquals2(element) {
  return element === 2;
});
console.log(result) // => 2
```

JS. Arrays

findIndex()

Метод findIndex() возвращает индекс первого найденного в массиве элемента, который подходит под условие переданной функции. Если же ни одного подходящего элемента не найдётся, то метод вернёт -1.

Если вам нужно получить элемент, а не его индекс, то используйте метод find(). А если необходимо проверить наличие чего-либо в массиве, то сначала обратите внимание на метод includes().

JS. Arrays

findIndex()

Синтаксис

array.findIndex(callback(currentValue, index, arr), thisArg)

Метод findIndex() не изменяет массив, для которого он был вызван.

JS. Arrays

sort()

Метод sort производит сортировку массива и возвращает уже измененный массив. Необязательным параметром можно указать собственную функцию для сортировки.

```
const arr = ['d', 'b', 'a', 'c'];

console.log(arr.sort());
```

Результат выполнения кода: ['a', 'b', 'c', 'd']

JS. Arrays

sort()

Порядок сортировки может быть буквенный, числовой, в порядке возрастания, или в порядке убывания.

По умолчанию метод sort() сортирует значения в виде строк в алфавитном порядке возрастания.

Примечание: Этот метод изменяет исходный массив (не создает новый массив, а производит сортировку переданного массива)

JS. Arrays

sort()

Синтаксис

array.sort(compareFunction)

Сортировка чисел в массиве в порядке возрастания:

```
const items = [30, 100, 1, 5, 27, 10];
 items.sort(function(a, b){return a-b}); // => 1,5,10,27,30,100
```

Сортировка чисел в массиве в порядке убывания:

```
const items = [30, 100, 1, 5, 27, 10];
 items.sort(function(a, b){return b-a}); // => 100,30,27,10,5,1
```

JS. Arrays

forEach()

Метод массива forEach() позволяет применить колбэк-функцию ко всем элементам массива. Можно использовать вместо классического цикла for. В отличие от него forEach() выглядит более читабельным и понятным.

```
const numbers = [1, 2, 3, 4]


numbers.forEach((num) => {
  const square = num * num
  console.log('Квадрат числа равен: ' + square)
})
```

JS. Arrays

forEach()

Метод в параметре получает функцию, которая выполнится для каждого элемента массива.

В эту функцию можно передавать три параметра. Если эти параметры есть (они не обязательны), то в первый автоматически попадет элемент массива, во второй попадет его номер в массиве (индекс), а в третий - сам массив.

Данный метод, в отличие от предыдущих, ничего не возвращает.

JS. Arrays

forEach()

Метод forEach() можно использовать, когда вам необходимо совершить одну и ту же операцию над всеми элементами массива.

Хотя в JavaScript уже есть возможность делать это, используя цикл for, метод forEach() — это отличная альтернатива с рядом преимуществ:

- Использование метода forEach() является декларативным способом обозначить нашу операцию. С точки зрения читабельности кода это больше приближено к естественному языку и лаконично.
- Позволяет удобно получать элемент в текущей итерации, без необходимости всякий раз обращаться к массиву по индексу.

Однако вместе с тем мы получаем и несколько недостатков:

- В forEach() не будут работать return, break и continue, а следовательно, мы никак не можем прервать или пропустить итерацию. Потому, если для решения задачи необходимо воспользоваться каким-либо из этих операторов, придётся использовать обычный цикл for.
- forEach() обрабатывает элементы массива в прямом порядке, то есть мы не можем пройти по массиву с конца.

JS. Arrays

map()

Метод map() позволяет трансформировать один массив в другой при помощи функций-колбэка. Переданная функция будет вызвана для каждого элемента массива по порядку. Из результатов вызова функции будет собран новый массив.

Метод .map() перебирает каждый элемент массива, производит с ним какие-либо действия и добавляет в новый массив, который мы получаем после окончания работы метода.

Новый массив, который создается в результате работы метода .map() всегда имеет такую же длину, что и исходный массив.

JS. Arrays

map()

```
const nums = [1, 2, 3, 4, 5, 6, 7, 8, 9]


const squares = nums.map(function (num) {
  return num * num
})


console.log(squares) // = > [1, 4, 9, 16, 25, 36, 49, 64, 81]
```

JS. Arrays

map()

Как и другим похожим методам, map() необходимо передать колбэк-функцию, которая будет возвращать какое-то значение. Именно это значение попадёт в итоговый трансформированный массив.

Функция, которую мы передаём в метод map(), может принимать три параметра:

item — элемент массива в текущей итерации;

index — индекс текущего элемента;

arr — сам массив, который мы перебираем.

JS. Arrays

map()

Часто возникают ситуации, когда на основе одного массива нужно создать другой массив, как-нибудь трансформировав исходные значения.

map() возвращает новый массив, при этом исходный массив никак не изменится.

При работе с map() необходимо возвращать значение из функции-колбэка. Если не вернуть значение — например, забыв обработать какую-то ветку условия, то в итоговом массиве будет undefined

При работе с React или другой похожей на неё библиотекой map() — это самый распространённый способ трансформировать массив данных в компоненты, которые в итоге будут на странице

JS. Arrays

filter()

Метод массива .filter() позволяет получить новый массив, отфильтровав элементы с помощью переданной колбэк-функции. Колбэк-функция будет вызвана для каждого элемента массива и по результату функции примет решение включать этот элемент в новый массив или нет.

Метод .filter() создает новый массив, куда добавляются все элементы исходного массива, которые соответствуют нашим условиям. Длина нового массива всегда отличается от длины исходного массива.

Каждый элемент исходного массива проверяется на соответствие нашим условиям. В случае соответствия, возвращается true и элемент добавляется в новый массив. В противном случае возвращается false и .filter переходит к оценке следующего элемента.

JS. Arrays

filter()

В новом массиве отфильтрованные элементы будут находиться в том же порядке, в котором они были в исходном массиве.

.filter() возвращает новый массив, при этом исходный массив никак не изменится.

```
const languages = ["Java", "TypeScript", "C#", "JavaScript"]


const jLanguages = languages.filter(function (language) {
  return language.startsWith("J")
})
```

JS. Arrays

filter()

Функция, которую мы передаём в метод .filter(), принимает три параметра:

item — элемент массива в текущей итерации;

index — индекс текущего элемента;

arr — сам массив, который мы перебираем.

```
const languages = ["Java", "TypeScript", "C#", "JavaScript"]

languages.filter(function (item, index, arr) {
  console.log("Текущий элемент " + item)
  console.log("Индекс " + index)
  console.log("Массив " + arr)
  return index >= 3
})
```

JS. Arrays

reduce()

Метод массива reduce() позволяет превратить массив в любое другое значение с помощью переданной функции-колбэка и начального значения. Функция-колбэк будет вызвана для каждого элемента массива, и всегда должна возвращать результат.

С помощью данного метода можно сохранять промежуточный результат и возвращать новые объекты, массивы или, например, числа.

JS. Arrays

reduce()

Находим сумму элементов:

```
const nums = [1, 2, 3, 4, 5, 6, 7, 8]

const sum = nums.reduce(function (currentSum, currentNumber) {
  return currentSum + currentNumber
}, 0) // => 36
```

JS. Arrays

reduce()

Метод reduce() принимает два параметра: функцию-колбэк и начальное значение для аккумулятора.

Сама функция-колбэк может принимать четыре параметра:

acc — текущее значение аккумулятора;

item — элемент массива в текущей итерации;

index — индекс текущего элемента;

arr — сам массив, который мы перебираем.

JS. Arrays

reduce()

```
const nums = [1, 2, 3, 4, 5, 6, 7, 8]

// Не забываем, что аккумулятор идет первым!

function findAverage(acc, item, index, arr) {

 const sum = acc + item

 // Если мы на последнем элементе, то вычисляем среднее арифметическое делением на кол-во элементов:

 if (index === arr.length - 1) {

  return sum / arr.length

 }

 return sum

}

const average = nums.reduce(findAverage, 0)

// 4.5
```

JS. Arrays

reduce()

Функция обязательно должна возвращать значение, поскольку в каждой следующей итерации значение в acc будет результатом, который вернулся на предыдущем шаге. Логичный вопрос, который может здесь возникнуть — какое значение принимает acc во время первой итерации? Им будет то самое начальное значение, которое передаётся вторым аргументом в метод reduce().

Начальное значение для аккумулятора можно не указывать явно. В таком случае на первой итерации аккумулятор будет равен значению первого элемента в массиве

```
const arr = [1, 2, 3]

const sum = arr.reduce(function (acc, val) {

  return acc + val

})

console.log(sum) // => 6
```

JS. Arrays

reduce()

Главной особенностью reduce(), которую важно запомнить, является наличие аккумулятора. Аккумулятор — это и есть то новое вычисляемое значение.

JS. Arrays

reduce()

Задача: вычислить сумму денег на всех счетах.

```
const bankAccounts = [ { id: "123", amount: 19 }, { id: "345", amount: 33 }, { id: "567", amount: 4 }, { id: "789", amount: 20 }]
const totalAmount = bankAccounts.reduce(
 // Аргумент sum является аккумулятором, в нём храним промежуточное значение
 function (sum, currentAccount) {
  // Каждую итерацию берём текущее значение и складываем его с количеством денег на текущем счету
  return sum + currentAccount.amount
 },
 0 // Начальное значение аккумулятора
)
console.log(totalAmount) // => 76
```