

JS: Arrays, loops

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)

Повторим;)

Какие математические операторы вы знаете?

Чем отличается строгое и нестрогое равенство

Расскажите синтаксис условного оператора

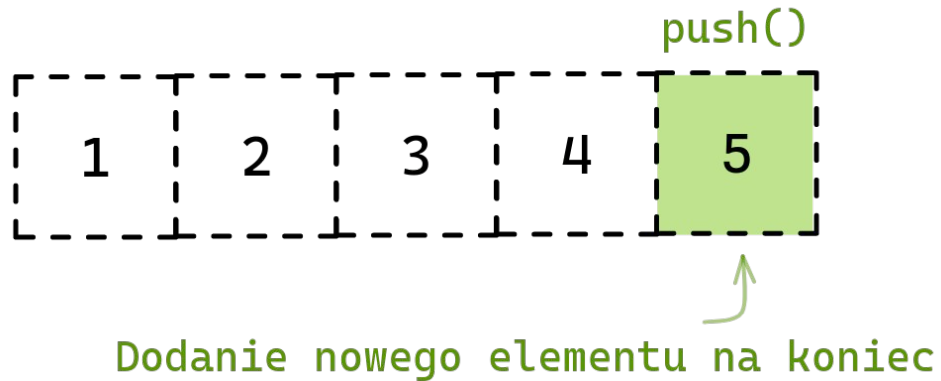
ЦЕЛЬ

Изучить особенности и синтаксис работы с массивами в js. Изучить методы push, pull, shift, unshift. Изучить циклы

ПЛАН ЗАНЯТИЯ

- Массивы: доступ к элементам, длина, добавление, удаление
- Циклы

Массивы



Довольно часто мы понимаем, что нам необходима упорядоченная коллекция данных, в которой присутствуют 1-й, 2-й, 3-й элементы и т.д.

Например, она понадобится нам для хранения списка чего-либо: пользователей, товаров, элементов HTML и т.д.

Для хранения упорядоченных коллекций существует особая структура данных, которая называется массив, Array.

Массив, состоящий из 5 элементов

123	7	50	-9	24
0	1	2	3	4
индексы элементов массива				

Объявление

Существует два варианта синтаксиса для создания пустого массива:

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

Практически всегда используется второй вариант синтаксиса. В скобках мы можем указать начальные значения элементов:

```
1 let fruits = ["Яблоко", "Апельсин", "Слива"];
```


Объявление

Элементы массива нумеруются, начиная с нуля.

Мы можем получить элемент, указав его номер в квадратных скобках:

```
1 let fruits = ["Яблоко", "Апельсин", "Слива"];  
2  
3 alert( fruits[0] ); // Яблоко  
4 alert( fruits[1] ); // Апельсин  
5 alert( fruits[2] ); // Слива
```

Мы можем заменить элемент:

```
1 fruits[2] = 'Груша'; // теперь ["Яблоко", "Апельсин", "Груша"]
```

...Или добавить новый к существующему массиву:

```
1 fruits[3] = 'Лимон'; // теперь ["Яблоко", "Апельсин", "Груша", "Лимон"]
```

Длина массива

Общее число элементов массива содержится в его свойстве `length`:

```
1 let fruits = ["Яблоко", "Апельсин", "Слива"];  
2  
3 alert( fruits.length ); // 3
```

Содержание массива

В массиве могут храниться элементы любого типа.

```
let mixedArray = [42, "Hello", true, [1, 2, 3]];
```

Методы pop/push, shift/unshift

Очередь – один из самых распространённых вариантов применения массива. В области компьютерных наук так называется упорядоченная коллекция элементов, поддерживающая два вида операций:

- **push** добавляет элемент в конец.
- **shift** удаляет элемент в начале, сдвигая очередь, так что второй элемент становится первым.



На практике необходимость в этом возникает очень часто. Например, очередь сообщений, которые надо показать на экране.

Методы pop/push, shift/unshift

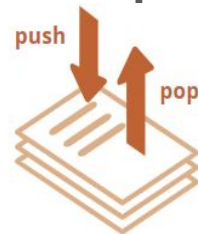
Существует и другой вариант применения для массивов – структура данных, называемая стек.

Она поддерживает два вида операций:

- push добавляет элемент в конец.
- pop удаляет последний элемент.

Таким образом, новые элементы всегда добавляются или удаляются из «конца».

Примером стека обычно служит колода карт: новые карты кладутся наверх и берутся тоже сверху:



Методы, работающие с концом массива:

pop

Удаляет последний элемент из массива и возвращает его:

```
1 let fruits = ["Яблоко", "Апельсин", "Груша"];
2
3 alert( fruits.pop() ); // удаляем "Груша" и выводим его
4
5 alert( fruits ); // Яблоко, Апельсин
```

Методы, работающие с концом массива:

push

Добавляет элемент в конец массива:

```
1  let fruits = ["Яблоко", "Апельсин"];  
2  
3  fruits.push("Груша");  
4  
5  alert( fruits ); // Яблоко, Апельсин, Груша
```


Методы, работающие с началом массива:

shift

Удаляет из массива первый элемент и возвращает его:

```
1 let fruits = ["Яблоко", "Апельсин", "Груша"];
2
3 alert( fruits.shift() ); // удаляем Яблоко и выводим его
4
5 alert( fruits ); // Апельсин, Груша
```

Методы, работающие с началом массива:

unshift

Добавляет элемент в начало массива:

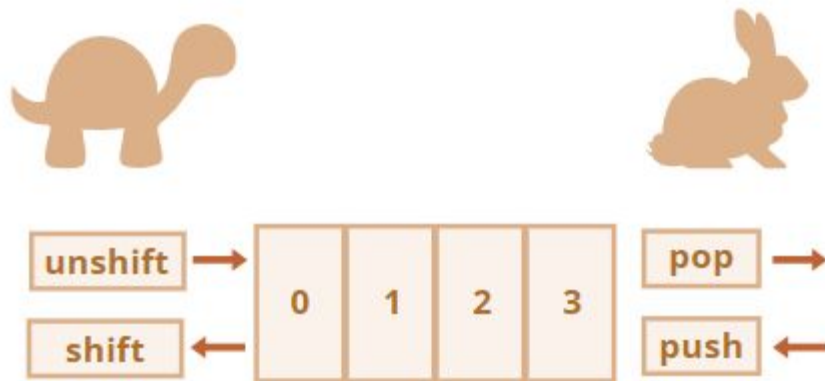
```
1 let fruits = ["Апельсин", "Груша"];
2
3 fruits.unshift('Яблоко');
4
5 alert( fruits ); // Яблоко, Апельсин, Груша
```

Методы `push` и `unshift` могут добавлять сразу несколько элементов:

```
1 let fruits = ["Яблоко"];
2
3 fruits.push("Апельсин", "Груша");
4 fruits.unshift("Ананас", "Лимон");
5
6 // ["Ананас", "Лимон", "Яблоко", "Апельсин", "Груша"]
7 alert( fruits );
```

Эффективность

Методы `push/pop` выполняются быстро, а методы `shift/unshift` – медленно.



Циклы



Цикл “for”

Более сложный, но при этом самый распространённый цикл — цикл for.

```
1  for (начало; условие; шаг) {  
2    // ... тело цикла ...  
3  }
```

Цикл “for”

Давайте разберёмся, что означает каждая часть, на примере. Цикл ниже выполняет `alert(i)` для `i` от 0 до (но не включая) 3:

```
1 for (let i = 0; i < 3; i++) { // выведет 0, затем 1, затем 2
2   alert(i);
3 }
```

Прерывание цикла: «break»

Обычно цикл завершается при вычислении условия в false.

Но мы можем выйти из цикла в любой момент с помощью специальной директивы **break**.

Например, следующий код подсчитывает сумму вводимых чисел до тех пор, пока посетитель их вводит, а затем – выдаёт:

```
1  let sum = 0;
2
3  while (true) {
4
5      let value = +prompt("Введите число", '');
6
7      if (!value) break; // (*)
8
9      sum += value;
10
11 }
12 alert( 'Сумма: ' + sum );
```


Переход к следующей итерации: continue

Директива continue – «облегчённая версия» break. При её выполнении цикл не прерывается, а переходит к следующей итерации (если условие все ещё равно true).

Её используют, если понятно, что на текущем повторе цикла делать больше нечего.

Например, цикл ниже использует continue, чтобы выводить только нечётные значения:

```
1 for (let i = 0; i < 10; i++) {  
2  
3   // если true, пропустить оставшуюся часть тела цикла  
4   if (i % 2 == 0) continue;  
5  
6   alert(i); // 1, затем 3, 5, 7, 9  
7 }
```

Перебор элементов массива - for

Одним из самых старых способов перебора элементов массива является цикл for по цифровым индексам:

```
1 let arr = ["Яблоко", "Апельсин", "Груша"];  
2  
3 for (let i = 0; i < arr.length; i++) {  
4     alert( arr[i] );  
5 }
```

Цикл while

Цикл while имеет следующий синтаксис:

```
1 while (condition) {  
2     // код  
3     // также называемый "телом цикла"  
4 }
```

Цикл while

Код из тела цикла выполняется, пока условие condition истинно.

Например, цикл ниже выводит *i*, пока $i < 3$:

```
1  let i = 0;
2  while (i < 3) { // выводит 0, затем 1, затем 2
3      alert( i );
4      i++;
5  }
```

Цикл while

Любое выражение или переменная может быть условием цикла, а не только сравнение: условие `while` вычисляется и преобразуется в логическое значение.

Например, `while (i)` – более краткий вариант `while (i != 0)`:

```
1  let i = 3;
2  while (i) { // когда i будет равно 0, условие станет ложным
3      alert( i );
4      i--;
5  }
```

Цикл “do...while”

Проверку условия можно разместить под телом цикла, используя специальный синтаксис do..while:

```
1  do {  
2    // тело цикла  
3  } while (condition);
```

Цикл “do...while”

Цикл сначала выполнит тело, а затем проверит условие `condition`, и пока его значение равно `true`, он будет выполняться снова и снова.

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

Такая форма синтаксиса оправдана, если вы хотите, чтобы тело цикла выполнилось **хотя бы один раз**

Перебор элементов массива - for...of

Оператор for...of выполняет цикл обхода

```
1 let iterable = [10, 20, 30]
2
3 for (let value of iterable) {
4   value += 1
5   console.log(value)
6 }
7 // 11
8 // 21
9 // 31
```


Методы работы со строками

Методы работы со строками

Доступ к элементам
массива по индексу

```
let text = "Hello, World!";
```

// Длина строки

```
let length = text.length; // 13
```

// Преобразование в верхний/нижний регистр

```
let upperCase = text.toUpperCase(); // "HELLO, WORLD!"
```

```
let lowerCase = text.toLowerCase(); // "hello, world!"
```

// Получение подстроки

```
let substring = text.substring(0, 5); // "Hello"
```

// Поиск подстроки

```
let indexOfWorld = text.indexOf("World"); // 7
```

Метод - split

Описание:

Метод split() разбивает строку на массив строк по заданному разделителю

Синтаксис:

```
str.split(separator, limit)
```

separator

Необязательный параметр. Указывает символы, используемые в качестве разделителя внутри строки.

limit

Необязательный параметр. Целое число, определяющее ограничение на количество найденных подстрок.

Метод - split

Возвращаемое значение:

Метод split() возвращает новый массив.

Примеры:

```
1 let names = 'Вася, Петя, Маша';
2
3 let arr = names.split(', ');
4
5 for (let name of arr) {
6   alert( `Сообщение получат: ${name}.` ); // Сообщение получат: Вася (и другие имена)
7 }
```

```
1 let arr = 'Вася, Петя, Маша, Саша'.split(', ', 2);
2
3 alert(arr); // Вася, Петя
```



Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH