



W



Web Academy
Programming Courses

Front-end с нуля

Тренер: Зинченко Андрей

Лекция 6

web-academy.com.ua

A

Содержание

W

1. Конструкция switch
2. Стрелочные функции
3. Рекурсия и стек
4. Массивы



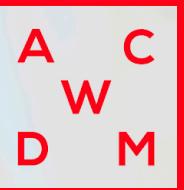
W

Switch



Конструкция switch заменяет собой сразу несколько if.
Она представляет собой более наглядный способ сравнить выражение сразу с несколькими вариантами.

Switch



```
let a = 2 + 2;

switch (a) {
    case 3:
        alert( 'Маловато' );
        break;
    case 4:
        alert( 'В точку!' );
        break;
    case 5:
        alert( 'Перебор' );
        break;
    default:
        alert( "Нет таких значений" );
}
```

Switch

A	C
W	
D	M

```
let arg = prompt("Введите число?");  
switch (arg) {  
    case '0':  
    case '1':  
        alert( 'Один или ноль' );  
        break;  
  
    case '2':  
        alert( 'Два' );  
        break;  
  
    case '3':  
        alert( 'Никогда не выполнится!' );  
        break;  
    default:  
        alert( 'Неизвестное значение' );  
}  
A
```

Arrow functions



Существует ещё более простой и краткий синтаксис для создания функций, который часто лучше, чем синтаксис Function Expression.

Он называется «функции-стрелки» или «стрелочные функции» (arrow functions), т.к. выглядит следующим образом:

```
let sum = (a, b) => a + b;
```



Рекурсия



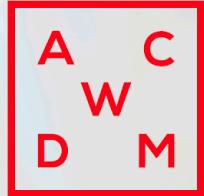
Рекурсия - это приём программирования, полезный в ситуациях, когда задача может быть естественно разделена на несколько аналогичных, но более простых задач. Или когда задача может быть упрощена до несложных действий плюс простой вариант той же задачи. Или, как мы скоро увидим, для работы с определёнными структурами данных.

В процессе выполнения задачи в теле функции могут быть вызваны другие функции для выполнения подзадач. Частный случай подвызова - когда функция вызывает *сама себя*. Это как раз и называется *рекурсией*.

W

Рекурсия

В качестве первого примера напишем функцию `pow(x, n)`, которая возводит x в натуральную степень n . Иначе говоря, умножает x на само себя n раз.



Рекурсия

A	C
W	
D	M

```
function pow(x, n) {  
    let result = 1;  
  
    // умножаем result на x n раз в цикле  
    for (let i = 0; i < n; i++) {  
        result *= x;  
    }  
  
    return result;  
}  
  
alert( pow(2, 3) ); // 8
```

A

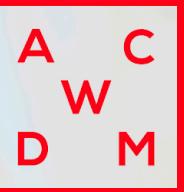
Рекурсия

A	C
W	
D	M

```
function pow(x, n) {  
    if (n == 1) {  
        return x;  
    } else {  
        return x * pow(x, n - 1);  
    }  
}  
  
alert( pow(2, 3) ); // 8
```

A

Рекурсия



Recursive

```
factorial(6)
6 * factorial(5)
6 * (5 * factorial(4))
6 * (5 * (4 * factorial(3)))
6 * (5 * (4 * (3 * factorial(2))))
6 * (5 * (4 * (3 * (2 * factorial(1))))))
6 * (5 * (4 * (3 * (2 * 1))))))
6 * (5 * (4 * (3 * 2))))
6 * (5 * (4 * 6))
6 * (5 * 24)
6 * 120
720
```

A large black curved arrow starts from the first line 'factorial(6)' and sweeps down to the final result '720', indicating the flow of the recursive computation.

W

Рекурсия



```
function pow(x, n) {  
    return (n == 1) ? x : (x * pow(x, n - 1));  
}
```



Массивы

Объекты позволяют хранить данные со строковыми ключами. Это замечательно.

Но 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.

В этом случае использовать объект неудобно, так как он не предоставляет методов управления порядком элементов. Мы не можем вставить новое свойство «между» уже существующими. Объекты просто не предназначены для этих целей.

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





Массивы

A C
W M
D

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



Массивы



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

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





Массивы



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



Массивы

A	C
W	
D	M

```
// разные типы значений
let arr = [ 'Яблоко', { name: 'Джон' }, true, function() { alert('привет'); } ];

// получить элемент с индексом 1 (объект) и затем показать его свойство
alert( arr[1].name ); // Джон

// получить элемент с индексом 3 (функция) и выполнить её
arr[3](); // привет
```



Многомерный массив



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

alert( matrix[1][1] ); // 5, центральный элемент
```



Перебор элементов



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

A



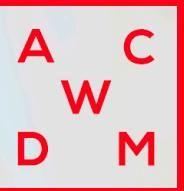
Перебор элементов

A	C
W	
D	M

```
let fruits = ["Яблоко", "Апельсин", "Слива";  
  
// проходит по значениям  
for (let fruit of fruits) {  
    alert( fruit );  
}
```

W

pop

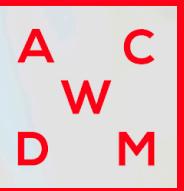


Удаляет последний элемент из массива и возвращает его:

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

W

push

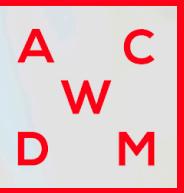


Добавляет элемент в конец массива:

```
let fruits = ["Яблоко", "Апельсин"];  
  
fruits.push("Груша");  
  
alert( fruits ); // Яблоко, Апельсин, Груша
```

W

shift



Удаляет из массива первый элемент и возвращает его:

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

unshift

A	C
W	
D	M

Добавляет элемент в начало массива:

```
let fruits = ["Апельсин", "Груша"];  
  
fruits.unshift('Яблоко');  
  
alert( fruits ); // Яблоко, Апельсин, Груша
```

push / unshift

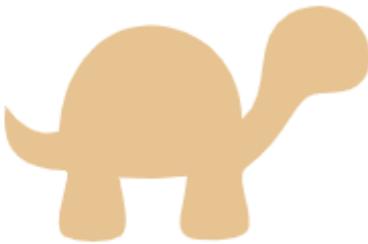
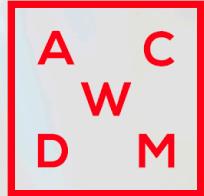


Методы `push` и `unshift` могут добавлять сразу несколько элементов:

```
let fruits = ["Яблоко"];
fruits.push("Апельсин", "Груша");
fruits.unshift("Ананас", "Лимон");
// ["Ананас", "Лимон", "Яблоко", "Апельсин", "Груша"]
alert( fruits );
```

W

Эффективность



unshift



shift



pop



push



delete(object)



```
let arr = ["I", "go", "home"];  
  
delete arr[1]; // удалить "go"  
  
alert( arr[1] ); // undefined  
  
// теперь arr = ["I", , "home"];  
alert( arr.length ); // 3
```

W

splice



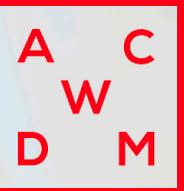
Метод `arr.splice(str)` - это универсальный «швейцарский нож» для работы с массивами. Умеет всё: добавлять, удалять и заменять элементы.

```
arr.splice(index[, deleteCount, elem1, ..., elemN])
```

A

W

splice



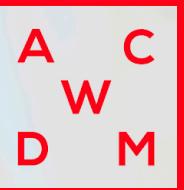
```
let arr = ["Я", "изучаю", "JavaScript"];
```

```
arr.splice(1, 1); // начиная с позиции 1, удалить 1 элемент
```

```
alert( arr ); // осталось ["Я", "JavaScript"]
```

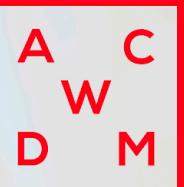
W

splice



```
let arr = ["Я", "изучаю", "JavaScript", "прямо", "сейчас";  
// удалить 3 первых элемента и заменить их другими  
arr.splice(0, 3, "Давай", "танцевать");  
  
alert( arr ) // теперь ["Давай", "танцевать", "прямо", "сейчас"]
```

splice



```
let arr = [1, 2, 5];
```

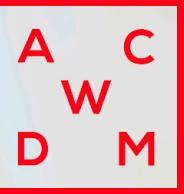
```
// начиная с индекса -1 (перед последним элементом)  
// удалить 0 элементов,
```

```
// затем вставить числа 3 и 4
```

```
arr.splice(-1, 0, 3, 4);
```

```
alert( arr ); // 1,2,3,4,5
```

slice



```
arr.slice([start], [end])
```

Он возвращает новый массив, в который копирует элементы, начиная с индекса start и до end (не включая end). Оба индекса start и end могут быть отрицательными. В таком случае отсчёт будет осуществляться с конца массива.

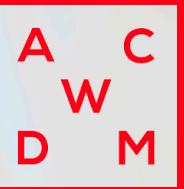
W

slice

A	C
W	
D	M

```
let arr = ["t", "e", "s", "t"];  
  
alert( arr.slice(1, 3) ); // e,s (копирует с 1 по 3)  
  
alert( arr.slice(-2) ); // s,t (копирует с -2 до конца)
```

concat



Метод `arr.concat` создаёт новый массив, в который копирует данные из других массивов и дополнительные значения.

```
let arr = [1, 2];

// создать массив из: arr и [3,4]
alert( arr.concat([3, 4]) ); // 1,2,3,4

// создать массив из: arr и [3,4] и [5,6]
alert( arr.concat([3, 4], [5, 6]) ); // 1,2,3,4,5,6

// создать массив из: arr и [3,4], потом добавить значения 5 и 6
alert( arr.concat([3, 4], 5, 6) ); // 1,2,3,4,5,6
```

W

forEach

A	C
W	
D	M

```
arr.forEach(function(item, index, array) {  
    // ... делать что-то с item  
});
```

```
["Bilbo", "Gandalf", "Nazgul"].forEach((item, index, array) => {  
    alert(` ${item} имеет позицию ${index} в ${array}`);  
});
```

WindexOf / lastIndexOf / includes

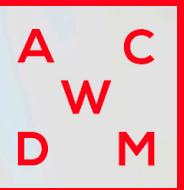


```
let arr = [1, 0, false];

alert( arr.indexOf(0) ); // 1
alert( arr.indexOf(false) ); // 2
alert( arr.indexOf(null) ); // -1

alert( arr.includes(1) ); // true
```

find / findIndex



Представьте, что у нас есть массив объектов. Как нам найти объект с определённым условием?

```
let users = [  
    {id: 1, name: "Вася"},  
    {id: 2, name: "Петя"},  
    {id: 3, name: "Маша"}  
];  
  
let user = users.find(item => item.id == 1);  
  
alert(user.name); // Вася
```

A



filter

A C
W M
D

Метод `find` ищет один (первый попавшийся) элемент, на котором функция-колбэк вернёт `true`.

```
let users = [  
  {id: 1, name: "Вася"},  
  {id: 2, name: "Петя"},  
  {id: 3, name: "Маша"}  
];
```

```
// возвращает массив, состоящий из двух первых пользователей  
let someUsers = users.filter(item => item.id < 3);  
  
alert(someUsers.length); // 2
```

W

map



Он вызывает функцию для каждого элемента массива и возвращает массив результатов выполнения этой функции.

```
let lengths = ["Bilbo", "Gandalf", "Nazgul"].map(item => item.length);
alert(lengths); // 5,7,6
```

W

Sort

A	C
W	
D	M

Он возвращает отсортированный массив, но обычно возвращаемое значение игнорируется, так как изменяется сам arr.

```
function compareNumeric(a, b) {  
    if (a > b) return 1;  
    if (a == b) return 0;  
    if (a < b) return -1;  
}  
  
let arr = [ 1, 2, 15 ];  
  
arr.sort(compareNumeric);  
  
alert(arr); // 1, 2, 15
```

A



split / join

A	C
W	
D	M



```
let names = 'Вася, Петя, Маша';

let arr = names.split(', ');

for (let name of arr) {
  alert(`Сообщение получат: ${name}.`); // Сообщение получат: Вася (и другие имена)
}

let arr = ['Вася', 'Петя', 'Маша'];

let str = arr.join(';'); // объединить массив в строку через ;

alert( str ); // Вася;Петя;Маша
```



W

reduce / reduceRight

A	C
W	
D	M

```
let value = arr.reduce(function(previousValue, item, index, array) {  
    // ...  
}, [initial]);
```

```
let arr = [1, 2, 3, 4, 5];
```

```
let result = arr.reduce((sum, current) => sum + current, 0);
```

```
alert(result); // 15
```