Удаление элементов массива

Убрать из массива последний элемент можно, добавив к его имени .pop(). Метод pop делает сразу два дела: удаляет последний элемент из массива и возвращает этот элемент в виде значения. Для примера начнем с нашего массива animals ["Белый медведь", "Мартышка", "Кот", "Пес", "Лама"]. Далее создадим новую переменную lastAnimal и сохраним в ней последний элемент, вызвав animals.pop().

```
animals;
["Белый медведь", "Мартышка", "Кот", "Пёс", "Лама"]
var lastAnimal = animals.pop();
lastAnimal;
"Лама"
animals;
["Белый медведь", "Мартышка", "Кот", "Пёс"]
animals.pop();
"Пёс"
animals;
["Белый медведь", "Мартышка", "Кот"]
animals.unshift(lastAnimal);
4
animals;
["Лама", "Белый медведь", "Мартышка", "Кот"]
```

При вызове animals.pop() первый раз последний элемент массива animals, "Лама", был возвращен и сохранен в переменной lastAnimal. Кроме того, элемент "Лама" был удален из массива, в котором после этого осталось четыре элемента. При следующем вызове animals.pop() был удален из массива и возвращен элемент "Пес", а элементов в массиве осталось всего три.

Вызвав animals.pop() для элемента "Пес", мы не сохранили это значение в переменной, и оно пропало. С другой стороны, элемент "Лама" был сохранен в переменной lastAnimal, чтобы при случае им можно было снова воспользоваться. С помощью unshift(lastAnimal) мы добавили "Лама" обратно, в начало массива. В итоге получился массив ["Лама", "Белый медведь", "Мартышка", "Кот"].

Методы push и pop хорошо друг друга дополняют, поскольку порой нужно работать только с концом массива. Вы можете добавить элемент в конец вызовом push, а потом, когда это понадобится, забрать его оттуда вызовом pop.

Чтобы удалить из массива первый элемент, вернув его значение, используйте .shift():

```
animals;
["Лама", "Белый медведь", "Мартышка", "Кот"]
var firstAnimal = animals.shift();
firstAnimal;
"Лама"
animals;
["Белый медведь", "Мартышка", "Кот"]
```

Метод animals.shift() работает аналогично animals.pop(), но элемент берется из начала массива. В начале этого примера массив animals имел вид ["Лама", "Белый медведь", "Мартышка", "Кот"]. Вызов .shift () вернул первый элемент, "Лама", который мы сохранили в переменной firstAnimal. Поскольку .shift() не только возвращает элемент, но и удаляет его, в массиве animals осталось лишь ["Белый медведь", "Мартышка", "Кот"].

Методы unshift и shift добавляют и удаляют элементы с начала массива — так же как push и pop добавляют и удаляют элементы с конца.

Объединение массивов

Чтобы «склеить» два массива, создав таким образом новый массив, используйте команду first Array.concat(otherArray). Метод concat создает массив, в котором элементы из firstArray будут расположены перед элементами из otherArray.

Пускай у нас есть два списка — список пушистых животных и список чешуйчатых животных — и мы хотим их объединить. Если поместить наших пушистых животных в массив furryAnimals, а чешуйчатых — в массив scalyAnimals, команда furryAnimals.concat(scalyAnimals) создаст новый массив, в начале которого будут элементы из первого массива, а в конце — из второго.

```
var furryAnimals = ["Альпака", "Кольцехвостый лемур", "Йети"];
var scalyAnimals = ["Удав", "Годзилла"];
var furryAndScalyAnimals = furryAnimals.concat(scalyAnimals);
furryAndScalyAnimals;
["Альпака", "Кольцехвостый лемур", "Йети", "Удав", "Годзилла"]
furryAnimals
["Альпака", "Кольцехвостый лемур", "Йети"]
scalyAnimals;
["Удав", "Годзилла"]
```

Хоть команда firstArray. concat (otherArray) и возвращает массив, содержащий все элементы из firstArray и otherArray, сами эти массивы остаются прежними. Запросив содержимое furryAnimals и scalyAnimals, мы видим, что массивы не изменились.

Объединение нескольких массивов

С помощью concat можно объединить больше чем два массива. Для этого укажите дополнительные массивы в скобках, разделив их запятыми:

```
var furryAnimals = ["Альпака", "Кольцехвостый лемур", "Йети"];
var scalyAnimals = ["Удав", "Годзилла"];
var featheredAnimals = ["Ара", "Додо"];
var allAnimals = furryAnimals.concat(scalyAnimals, featheredAnimals);
allAnimals;
["Альпака", "Кольцехвостый лемур", "Йети", "Удав", "Годзилла", "Ара", "Додо"]
```

Мы видим, что пернатые животные из массива featheredAnimals оказались в самом конце нового массива, поскольку featheredAnimals был указан последним в скобках метода concat.

Поиск индекса элемента в массиве

Чтобы выяснить, какой у определенного элемента индекс в массиве, используйте .indexOf("элемент"). Создадим массив colors с названиями цветов, а затем получим индексы элементов «синий» и «зеленый» с помощью команд colors.indexOf("синий") и colors.indexOf("зеленый"). Поскольку "синий" располагается по индексу 2, colors.indexOf("синий") вернет 2. А "зеленый" находится по индексу 1, так что colors.indexOf("зеленый") вернет 1.

```
var colors = ["красный", "зелёный", "синий"];
colors.indexOf("синий");
2
colors.indexOf("зелёный");
1
```

Если элемента, индекс которого вы запрашиваете, в массиве нет, JavaScript вернет значение -1.

```
colors.indexOf("фиолетовый");
-1
```

Таким образом JavaScript сообщает: «элемент не найден», так или иначе возвращая из метода число.

Если элемент встречается в массиве больше чем один раз, indexOf вернет индекс того элемента, который находится ближе к началу массива.

```
var insects = ["Пчела", "Муравей", "Пчела", "Пчела", "Муравей"];
undefined
insects.indexOf("Пчела");
0
```

Превращаем массив в строку

Воспользовавшись методом .join(), можно соединить все элементы массива в одну большую строку.

```
var boringAnimals = ["Мартышка", "Кот", "Рыба", "Ящерица"];
boringAnimals.join();
"Мартышка,Кот,Рыба,Ящерица"
```

Метод join возвращает строку, в которой через запятую перечислены все элементы массива boringAnimals. Но что если мы не хотим использовать в качестве разделителя запятую?

Нам поможет метод .join ("разделитель"), который делает все то же самое, но вместо запятых ставит между элементами выбранный разделитель. Давайте попробуем три разных разделителя: дефис с пробелами по сторонам, звездочку * и союз «и» с пробелами по сторонам. Обратите внимание: разделитель нужно записывать в кавычках — ведь это строка.

```
var boringAnimals = ["Мартышка", "Кот", "Рыба", "Ящерица"];
boringAnimals.join(" - ");
"Мартышка - Кот - Рыба - Ящерица"
boringAnimals.join("*");
"Мартышка*Кот*Рыба*Ящерица"
boringAnimals.join(" и ");
"Мартышка и Кот и Рыба и Ящерица"
```

Этот вариант join удобен, когда у вас есть массив, из которого нужно сделать строку. Если же в массиве хранятся нестроковые значения, JavaScript преобразует их в строки перед тем, как соединить:

```
var myArray = [11, true, 14, 88];
myArray.join(" ");
"11 true 14 88"
```

Что полезного можно сделать с массивами

Теперь вы умеете разными способами создавать массивы и знаете немало действий с ними. Но как все это может вам пригодиться в жизни? В этом разделе мы разберем несколько коротких программ, посвященных практическому использованию массивов.

Поиск дороги домой

Представьте, что ваша подруга побывала у вас в гостях, а теперь хочет показать вам свой дом. Но вот незадача — вы никогда не бывали у нее раньше, а путь назад вам предстоит проделать в одиночку.

К счастью, вам в голову приходит хитрый способ решения этой проблемы: по дороге к дому подруги вы будете записывать возможные ориентиры (телефонную будку, вывеску магазина или аптеки, школу и т. д.). А по дороге назад, двигаясь по списку с конца, вычеркивать каждый встреченный ориентир — так вы всегда будете знать, куда идти дальше.

Построение массива с помощью push

Давайте напишем код для выполнения этих действий. Начнем с создания массива — пустого, поскольку, пока вы еще не отправились в гости, неизвестно, какие ориентиры вам повстречаются. Затем, по дороге к дому вашей подруги, мы будем добавлять описание каждого ориентира в массив с помощью push. И наконец, когда придет время идти домой, будем методом pop изымать каждый пройденный ориентир из массива.

```
var landmarks = [];
landmarks.push("Мой дом");
landmarks.push("Дорожка к дому");
landmarks.push("Мигающий фонарь");
landmarks.push("Протекающий гидрант");
landmarks.push("Пожарная станция");
landmarks.push("Приют для кошек");
landmarks.push("Моя бывшая школа");
landmarks.push("Дом подруги");
```

Здесь мы создали пустой массив landmarks и методом push сохранили в нем все ориентиры, замеченные по дороге к дому подруги.

Движемся в обратном порядке с помощью pop

Вы добрались до дома подруги, и можно изучить массив ориентиров. Разумеется, первым стоит "Мой дом", потом "Дорожка к дому", и т. д. до конца массива, где находится элемент "Дом подруги". Теперь, когда наступит время идти домой, вам останется лишь изымать из массива по одному элементу, и всегда будет понятно, куда идти дальше.

```
landmarks.pop();
"Дом подруги"
landmarks.pop();
"Моя бывшая школа"
landmarks.pop();
"Приют для кошек"
landmarks.pop();
"Пожарная станция"
landmarks.pop();
"Протекающий гидрант"
landmarks.pop();
"Мигающий фонарь"
landmarks.pop();
"Дорожка к дому"
landmarks.pop();
"Мой дом"
```

Вот вы и дома!

Заметили, что первый ориентир, который вы поместили в массив методом push, оказался также последним, который вы извлекли методом pop? А последний добавленный ориентир оказался первым извлеченным? Может показаться, что лучше бы первый добавленный элемент и извлекался всегда первым, однако извлекать элементы в обратном порядке в некоторых случаях удобно.

Такой подход нередко используется в больших программах — именно поэтому push и pop в JavaScript всегда под рукой.

Среди программистов такой способ работы с элементами называется «стек». Представьте, что стек—это стопка блинов. Всякий раз, когда готов новый блин, его кладут сверху стопки (как метод push), и, когда вы берете блин, чтобы его съесть, вы тоже берете его сверху (как метод pop). Снятие элементов со стека похоже на путешествие назад во времени: последним изымается элемент, который был в стеке первым. То же происходит с блинами: последний блин, который вы съедите, — это первый, который был приготовлен. На сленге программистов этот способ называется «последним вошел, первым вышел», английская аббревиатура LIFO (last in, first out). Есть и альтернативный подход — «первым вошел, первым вышел», аббревиатура FIFO (first in, first out). Его также называют очередью, поскольку таким же образом устроены очереди — первый человек, вставший в очередь, будет первым, которого обслужат.

Случайный выбор

Используя массивы, можно написать программу, которая выдает случайные варианты из заданного списка (наподобие «шара судьбы»). Однако сначала нужно разобраться, откуда нам брать случайные числа.

Использование Math.random()

Случайные числа можно генерировать с помощью специального метода Math.random(), который при каждом вызове возвращает случайное число от 0 до 1:

```
Math.random();
0.9298871867358685
Math.random();
0.8295785726513714
Math.random();
0.07565958867780864
```

Важно помнить, что Math.random() всегда возвращает число меньше 1, то есть никогда не возвращает собственно 1.

Если вам нужно число побольше, просто умножьте полученное из метода Math.random() значение на подходящий коэффициент. Например, если нужно случайное число от 0 до 10, умножьте Math. random() на 10:

```
Math.random() * 10;
7.582776751369238
Math.random() * 10;
0.9281923761591315
Math.random() * 10;
2.9693317296914756
```

Округление с помощью Math.floor()

И все же эти случайные значения нельзя использовать как индексы в массиве, поскольку индексы должны быть целыми числами, а не десятичными дробями. Чтобы исправить этот недостаток, нужен метод Math.floor(), округляющий число до ближайшего снизу целого значения (по сути, он просто отбрасывает все знаки после запятой).

```
Math.floor(9.296249609906226);
9
Math.floor(0.9281923761591315);
0
Math.floor(5.226786588318646);
5
```

Давайте используем оба метода, чтобы получить случайный индекс. Нужно лишь умножить Math.random() на длину массива и затем округлить полученное число методом Math.floor(). Например, если в массиве четыре элемента, это можно сделать так:

```
Math.floor(Math.random() * 4);
2 // может выпасть 0, 1, 2 или 3
```

При каждом запуске этот код будет возвращать случайное число от 1 до 3 (включая 0 и 3). Поскольку Math.random() всегда возвращает значение меньше 1, Math.random() * 4 никогда не вернет 4 или большее число.

Используя это случайное число как индекс, можно получить случайный элемент массива:

```
var randomWords = ["Взрыв", "Пещера", "Принцесса", "Карандаш"];
var randomIndex = Math.floor(Math.random() * 4);
randomWords[randomIndex];
"Принцесса"
```

С помощью Math.floor(Math.random() * 4); мы получили случайное число от 0 до 3. Сохранив это число в переменной randomIndex, мы использовали его как индекс для получения строки из массива randomWords.

В сущности, можно сделать этот код короче, избавившись от переменной randomIndex:

```
randomWords[Math.floor(Math.random() * 4)];
"Пещера"
```

Программа случайного выбора вариантов

Теперь давайте создадим массив с фразами, чтобы случайным образом выбирать их с помощью написанного ранее кода. Это и будет наш компьютерный «шар судьбы»! В комментариях указаны примеры вопросов, которые можно задать нашей программе.

```
var phrases = ["Звучит неплохо",
"Да, это определенно надо сделать",
"Не думаю, что это хорошая идея",
"Может, не сегодня?",
"Компьютер говорит нет"
];
// Мне съесть еще два-три пирожка с капустой?
phrases[Math.floor(Math.random() * 5)];
"Не думаю, что это хорошая идея"
// Мне пора писать программы к зачёту?
```

```
phrases[Math.floor(Math.random() * 5)];
"Да, это определенно надо сделать"
```

Мы создали массив phrases, в котором хранятся различные советы. Теперь, придумав вопрос, можно запросить случайный элемент из массива phrases, и полученный совет поможет принять решение!

Обратите внимание: поскольку в массиве с советами пять элементов, мы умножаем Math.random() на 5. Таким образом, мы всегда получим одно из пяти значений индекса: 0,1, 2, 3 или 4.

Генератор случайных дразнилок

Можно усовершенствовать код выбора вариантов, создав программу, которая при каждом запуске генерирует случайную дразнилку!

```
var randomBodyParts = ["глаз", "нос", "череп", "гузно", "нога", "чёлка"];
var randomAdjectives = ["вонючая", "поганая", "смердящая",
                        "унылая", "дурацкая", "паскудная"];
var randomWords = ["коровья лепёшка", "глиста", "рыба", "свинья", "метла", "жвачка"];
// Выбор случайной части тела из массива randomBodyParts:
var randomBodyPart = randomBodyParts[Math.floor(Math.random() * randomBodyParts.length)];
// Выбор случайного прилагательного из массива randomAdjectives:
var randomAdjective = randomAdjectives[Math.floor(Math.random() * randomAdjectives.length)];
// Выбор случайного слова из массива randomWords:
var randomWord = randomWords[Math.floor(Math.random() * randomWords.length)];
// Соединяем случайные строки в предложение:
var randomInsult = "У тебя " + randomBodyPart + " словно " +
                   randomAdjective + " " + randomWord + "!!!";
randomInsult;
"У тебя чёлка словно поганая метла!!!"
```

У нас есть три массива со словами, и мы с помощью трех индексов берем из каждого массива по случайному слову. Затем мы склеиваем их, помещая результат в переменную randomInsult, — это и есть готовая дразнилка. Обратите внимание, что мы добавили между randomAdjective и randomWord строку с единственным пробелом. Запустите этот код несколько раз при каждом запуске получится новая случайная дразнилка!

Объекты

Объекты JavaScript очень похожи на массивы, но для доступа к элементам объектов используются строки, а не числа. Эти строки называют ключами, или свойствами, а элементы, которые им соответствуют, — значениями. Вместе эти фрагменты информации образуют пары «ключ-значение». Причем если массивы используются главным образом как списки, хранящие множество элементов, то объекты часто применяют как одиночные сущности с множеством характеристик, или атрибутов. Например, ранее мы создали несколько массивов, хранящих названия разных животных. Но что если нужно хранить набор различных сведений об одном конкретном животном?

Создание объектов

Для хранения всевозможной информации об одном животном подойдет JavaScript-объект. Вот пример объекта, где хранятся сведения о кошке по имени Лира.

```
var cat = {
"legs": 4,
"name": "Лира",
"color": "Шоколадный"
};
```

Мы создали переменную под названием cat и присвоили ей объект с тремя парами «ключ-значение» (лапы, имя, окрас). При создании объекта используются фигурные скобки {} вместо квадратных, к которым мы привыкли, создавая массивы. Внутри фигурных скобок можно вводить пары «ключ-значение», а вместе скобки и пары значений называются литералом объекта. Литерал объекта — это быстрый способ создания объекта вместе с его содержимым.

При создании объекта ключ записывается перед двоеточием (:), а значение — после. Это двоеточие напоминает знак «равно», поскольку значения, стоящие слева, присваиваются именам (ключам), стоящим справа, что похоже на создание переменных со значениями. Все пары «ключ-значение» должны быть разделены запятыми — в нашем примере эти запятые стоят в конце строк. И обратите внимание, что после завершающей пары «ключ-значение» ("color": " Шоколадный") запятую ставить не нужно — следом за этой парой ставится закрывающая фигурная скобка.

Ключи без кавычек

Создавая первый объект, мы писали имена ключей в кавычках, однако это не обязательно. Следующая запись тоже является допустимым литералом объекта:

```
var cat = {
legs: 4,
name: "Лира",
color: "Шоколадный"
};
```

JavaScript знает, что ключи всегда строковые, поэтому можно обходиться без кавычек. В этом случае имена ключей должны соответствовать тем же правилам, что и имена переменных: например, в них не должно быть пробелов. Но если ключ указан в кавычках, пробелы в его имени допустимы:

```
var cat = {
legs: 4,
"full name": "Уси-Пусечка Лира Барсиковна",
color: "Шоколадный"
};
```

Помните, что, хотя ключ всегда является строковым (в кавычках он записан или без), значение, соответствующее этому ключу, может быть любого типа — даже переменной, в которой хранятся данные.

Кроме того, весь объект можно записать одной строкой, хотя читать такую программу будет, пожалуй, не слишком удобно:

```
var cat = {legs: 4, name: "Лира", color: "Шоколадный"};
```

Доступ к значениям внутри объектов

Хранящиеся в объектах значения можно получить с помощью квадратных скобок — так же, как элементы массива. Единственное различие в том, что вместо индекса (число) используется ключ (строка).

```
cat["name"];
"Лира"
```

Точно так же, как необязательны кавычки при записи литерала объекта, их можно опускать и при доступе к значениям по ключу. Однако в этом случае код будет немного другим:

```
cat.name;
"Лира"
```

Такую запись называют точечной нотацией. Вместо того чтобы писать имя ключа в кавычках внутри квадратных скобок, мы просто ставим точку, после которой пишем имя ключа, без кавычек. И, аналогично ключам без кавычек при записи литерала, такой прием сработает, только если ключ не содержит специальных символов — например, пробелов.

Теперь предположим, что вы хотите узнать, какие вообще ключи есть у данного объекта. Для этого в JavaScript есть удобное средство — команда Object.keys():

```
var dog = { name: "Олли", age: 6, color: "белый", bark: "Гав тяф тяф!"};
var cat = { name: "Гармония", age: 8, color: "шоколадный"};
Object.keys(dog);
["name", "age", "color", "bark"]
Object.keys(cat);
["name", "age", "color"]
```

Object.keys(anyObject) возвращает массив, содержащий все ключи объекта anyObject.

Добавление элементов объекта

Пустой объект похож на пустой массив, только вместо квадратных скобок при его создании используются фигурные:

```
var object = {};
```

Добавлять элементы объекта можно так же, как элементы массива, — но используя строки вместо чисел:

```
var cat = {};
cat["legs"] = 4;
cat["name"] = "Лира";
cat["color"] = "Шоколадный";
cat;
{legs: 4, name: "Лира", color: "Шоколадный"}
```

Мы начали с пустого объекта под названием cat, а затем поочередно добавили к нему три пары «ключ-значение». Потом мы ввели cat;, и браузер отобразил содержимое объекта. Тут надо отметить, что разные браузеры могут показывать объекты по-разному. Например, Chrome выводит объект cat в консоли следующим образом:

```
Object {legs: 4, name: "Лира", color: "Шоколадный"}
```

Chrome перечисляет ключи в таком порядке— (legs, name, color), но другие браузеры могут выводить их в другой очередности. Дело в том, что JavaScript хранит ключи объектов, не упорядочивая их.

В массивах элементы расположены строго один за другим: индекс 0 перед индексом 1, индекс 3 после индекса 2; однако в случае объектов неясно, как расположить элементы друг относительно друга. Должен ли ключ color стоять перед legs или после? «Правильного» ответа на этот вопрос нет, поэтому объекты хранят свои ключи без конкретной очередности, в результате чего разные браузеры показывают ключи в разном порядке. Так что никогда не полагайтесь в своих программах на тот или иной порядок ключей.

Добавление ключей через точку

Новые ключи также можно добавлять через точечную нотацию. Давайте перепишем этим способом предыдущий пример, то есть создадим пустой объект и заполним его данными:

```
var cat = {};
cat.legs = 4;
cat.name = "Лира";
cat.color = "Шоколадный";
```

Если обратиться к несуществующему свойству объекта, JavaScript вернет специальное значение undefined, сообщая таким образом: «здесь ничего нет». Например:

```
var dog = {
name: "Олли",
legs: 4,
isAwesome : true
};
dog.isBrown;
undefined
```

Здесь мы определили три свойства объекта dog: name, legs и isAwesome. Свойства isBrown среди них нет, поэтому dog.isBrown возвращает undefined.

Массивы объектов

До этого момента мы рассматривали только массивы и объекты, в которых содержатся данные простых типов, такие как числа и строки. Однако ничто не мешает сделать элементом массива или объекта другой массив или объект. Например, так может выглядеть массив с объектами, описывающими динозавров:

```
var dinosaurs = [
{name: "Тираннозавр рекс", period: "Верхнемеловой"},
{name: "Стегозавр", period: "Верхнеюрский"},
```

{name: "Платеозавр", period: "Триасовый"}

];

Получить сведения о первом динозавре можно уже известным нам способом — указав индекс в квадратных скобках:

dinosaurs[0];

{name: "Тираннозавр рекс", period: "Верхнемеловой"}

А если нужно только название первого динозавра, достаточно указать ключ объекта в еще одних квадратных скобках, следом за индексом:

dinosaurs[0]["name"];

"Тираннозавр рекс"

Другой вариант — воспользоваться точечной нотацией:

dinosaurs[0].name;

"Тираннозавр рекс"

Давайте рассмотрим более сложный пример — массив объектов со сведениями о друзьях, где в каждый из объектов вложено по еще одному массиву. Сначала создадим объекты, а затем поместим их в массив.

var anna = {name: "Анна", age: 18, luckyNumbers: [2, 4, 8, 16]};

var azamat = {name: "Азамат", age: 19, luckyNumbers: [3, 9, 40]};

var lenara = {name: "Ленара", age: 18, luckyNumbers: [1, 3, 7]};

Мы создали три объекта, сохранив их в переменных anna, azamat и lenara. У каждого из этих объектов есть по три свойства: name, age и luckyNumbers. Каждому ключу name соответствует строковое значение, ключу age — числовое, а ключу luckyNumbers — массив, содержащий несколько чисел.

Теперь создадим массив друзей:

var friends = [anna, azamat, lenara];

Итак, в переменной friends находится массив с тремя элементами: anna, azamat и lenara (каждый из них является объектом). Мы можем получить любой из объектов по его индексу в массиве:

friends[2];

{name: "Ленара", age: 18, luckyNumbers: Array[3]}

Здесь мы извлекли из массива третий объект, lenara (по индексу 2). Вместо массива luckyNumbers Chrome напечатал Array[3], что означает «это массив с тремя элементами». Также мы можем получить значение, хранящееся в объекте, указав индекс объекта в квадратных скобках, поставив точку и написав соответствующий ключ:

friends[0].name;

"Анна"

Этот код запрашивает элемент по индексу 0 (что соответствует переменной anna), а затем — свойство этого объекта, хранящееся по ключу "name" (это "Анна"). Можно даже получить значение из массива, находящегося в объекте, который, в свою очередь, находится в массиве friends:

friends[0].luckyNumbers[1];

4