Структуры данных

Оглавление

[Операторы и операнды 0](#_Toc447573385)

[Приоритет операций 1](#_Toc447573386)

[Побитовые операторы 6](#_Toc447573387)

[Используемая литература 7](#_Toc447573388)

# Введение в методы и свойства

Все значения в JavaScript, за исключением null и undefined, содержат набор вспомогательных функций и значений, доступных «через точку».

Такие функции называют «методами», а значения – «свойствами». Здесь мы рассмотрим основы использования свойств и методов.

У строки есть *свойство* length, содержащее длину:

|  |
| --- |
| alert( "Привет, мир!".length ); // 12  // Можно и записать строку в переменную, а потом запросить её свойство  var str = "Привет, мир!";  alert( str.length ); // 12 |

Также у строк есть *метод* toUpperCase(), который возвращает строку в верхнем регистре:

|  |
| --- |
| var hello = "Привет, мир!";  alert( hello.toUpperCase() ); // "ПРИВЕТ, МИР!" |

Есть методы и у чисел, например num.toFixed(n). Он округляет число num до n знаков после запятой, при необходимости добивает нулями до данной длины и возвращает в виде строки (удобно для форматированного вывода):

|  |
| --- |
| var n = 12.345;  alert( n.toFixed(2) ); // "12.35"  alert( n.toFixed(0) ); // "12"  alert( n.toFixed(5) ); // "12.34500" |

# Числа

Все числа в JavaScript, как целые так и дробные, имеют тип Number и хранятся в 64-битном формате[IEEE-754](http://en.wikipedia.org/wiki/IEEE_754-1985), также известном как «double precision».

Здесь мы рассмотрим различные тонкости, связанные с работой с числами в JavaScript.

В JavaScript можно записывать числа не только в десятичной, но и в шестнадцатеричной (начинается с 0x) системе счисления:

|  |
| --- |
| alert( 0xFF ); // 255 в шестнадцатиричной системе |

Также доступна запись в *«научном формате»* (ещё говорят «запись с плавающей точкой»), который выглядит как <число>e<кол-во нулей>.

Например, 1e3 – это 1 с 3 нулями, то есть 1000.

|  |
| --- |
| // еще пример научной формы: 3 с 5 нулями  alert( 3e5 ); // 300000 |

Если количество нулей отрицательно, то число сдвигается вправо за десятичную точку, так что получается десятичная дробь:

|  |
| --- |
| // здесь 3 сдвинуто 5 раз вправо, за десятичную точку.  alert( 3e-5 ); // 0.00003 <-- 5 нулей, включая начальный ноль |

Представьте, что вы собираетесь создать новый язык… Люди будут называть его «JavaScript» (или «LiveScript»… неважно).

Что должно происходить при попытке деления на ноль?

Как правило, ошибка в программе… Во всяком случае, в большинстве языков программирования это именно так.

Но создатель JavaScript решил пойти математически правильным путем. Ведь чем меньше делитель, тем больше результат. При делении на очень-очень маленькое число должно получиться очень большое. В математическом анализе это описывается через [пределы](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D0%B5%D0%BB_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), и если подразумевать предел, то в качестве результата деления на 0 мы получаем «бесконечность», которая обозначается символом ∞ или, в JavaScript: "Infinity".

|  |
| --- |
| alert( 1 / 0 ); // Infinity  alert( 12345 / 0 ); // Infinity |

**Infinity – особенное численное значение, которое ведет себя в точности как математическая бесконечность ∞.**

* Infinity больше любого числа.
* Добавление к бесконечности не меняет её.

|  |
| --- |
| alert( 1 / 0 ); // Infinity  alert( 12345 / 0 ); // Infinity |

**Бесконечность можно присвоить и в явном виде: var x = Infinity.**

Бывает и минус бесконечность -Infinity:

|  |
| --- |
| alert( -1 / 0 ); // -Infinity |

Бесконечность можно получить также, если сделать ну очень большое число, для которого количество разрядов в двоичном представлении не помещается в соответствующую часть стандартного 64-битного формата, например:

|  |
| --- |
| alert( 1e500 ); // Infinity |

Если математическая операция не может быть совершена, то возвращается специальное значениеNaN (Not-A-Number).

Например, деление 0/0 в математическом смысле неопределено, поэтому его результат NaN:

|  |
| --- |
| alert( 0 / 0 ); // NaN |

Значение NaN используется для обозначения математической ошибки и обладает следующими свойствами:

* Значение NaN – единственное, в своем роде, которое *не равно ничему, включая себя*.

Следующий код ничего не выведет:

|  |
| --- |
| if (NaN == NaN) alert( "==" ); // Ни один вызов  if (NaN === NaN) alert( "===" ); // не сработает |

Значение NaN можно проверить специальной функцией isNaN(n), которая преобразует аргумент к числу и возвращает true, если получилось NaN, и false – для любого другого значения.

|  |
| --- |
| var n = 0 / 0;  alert( isNaN(n) ); // true  alert( isNaN("12") ); // false, строка преобразовалась к обычному числу 12 |

Значение NaN «прилипчиво». Любая операция с NaN возвращает NaN.

|  |
| --- |
| alert( NaN + 1 ); // NaN |

Если аргумент isNaN – не число, то он автоматически преобразуется к числу.

# Строки

В JavaScript любые текстовые данные являются строками. Не существует отдельного типа «символ», который есть в ряде других языков.

Внутренним форматом строк, вне зависимости от кодировки страницы, является Юникод (Unicode).

Строки создаются при помощи двойных или одинарных кавычек:

|  |
| --- |
| var text = "моя строка";  var anotherText = 'еще строка';  var str = "012345"; |

В JavaScript нет разницы между двойными и одинарными кавычками.

Строки могут содержать специальные символы. Самый часто используемый из таких символов – это «перевод строки».

Он обозначается как \n, например:

|  |
| --- |
| alert( 'Привет\nМир' ); // выведет "Мир" на новой строке |

Есть и более редкие символы, вот их список:

| Специальные символы | |
| --- | --- |
| **Символ** | **Описание** |
| \b | Backspace |
| \f | Form feed |
| \n | New line |
| \r | Carriage return |
| \t | Tab |
| \uNNNN | Символ в кодировке Юникод с шестнадцатеричным кодом `NNNN`. Например, `\u00A9` -- юникодное представление символа копирайт © |

# Объекты

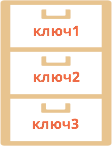
Объекты в JavaScript сочетают в себе два важных функционала.

Первый – это ассоциативный массив: структура, пригодная для хранения любых данных. В этой главе мы рассмотрим использование объектов именно как массивов.

Второй – языковые возможности для объектно-ориентированного программирования. Эти возможности мы изучим в последующих разделах учебника.

[Ассоциативный массив](http://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%BE%D1%86%D0%B8%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2) – структура данных, в которой можно хранить любые данные в формате ключ-значение.

Её можно легко представить как шкаф с подписанными ящиками. Все данные хранятся в ящичках. По имени можно легко найти ящик и взять то значение, которое в нём лежит.



В отличие от реальных шкафов, в ассоциативный массив можно в любой момент добавить новые именованные «ящики» или удалить существующие. Далее мы увидим примеры, как это делается.

Кстати, в других языках программирования такую структуру данных также называют *«словарь»* и *«хэш»*.

Пустой объект («пустой шкаф») может быть создан одним из двух синтаксисов:

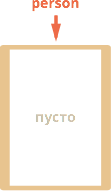
|  |
| --- |
| 1. o = new Object();  2. o = {}; // пустые фигурные скобки |

Обычно все пользуются синтаксисом (2), т.к. он короче.

Объект может содержать в себе любые значения, которые называются *свойствами объекта*. Доступ к свойствам осуществляется по *имени свойства* (иногда говорят *«по ключу»*).

Например, создадим объект person для хранения информации о человеке:

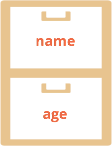
|  |
| --- |
| var person = {}; // пока пустой |



Основные операции с объектами – это создание, получение и удаление свойств.

Для обращения к свойствам используется запись «через точку», вида объект.свойство, например:

|  |
| --- |
| // при присвоении свойства в объекте автоматически создаётся "ящик"  // с именем "name" и в него записывается содержимое 'Вася'  person.name = 'Вася';  person.age = 25; // запишем ещё одно свойство: с именем 'age' и значением 25 |



Значения хранятся «внутри» ящиков. Обратим внимание – любые значения, любых типов: число, строка – не важно.

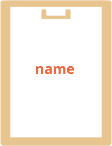
Чтобы прочитать их – также обратимся через точку:

|  |
| --- |
| alert( person.name + ': ' + person.age ); // "Вася: 25" |

Удаление осуществляется оператором delete:

|  |
| --- |
| delete person.age; |

Осталось только свойство name:



Иногда бывает нужно проверить, есть ли в объекте свойство с определенным ключом.

Для этого есть особый оператор: "in".

Его синтаксис: "prop" in obj, причем имя свойства – в виде строки, например:

|  |
| --- |
| if ("name" in person) {  alert( "Свойство name существует!" );  } |

Существует альтернативный синтаксис работы со свойствами, использующий квадратные скобкиобъект['свойство']:

|  |
| --- |
| var person = {};  person['name'] = 'Вася'; // то же что и person.name = 'Вася' |

Записи person['name'] и person.name идентичны, но квадратные скобки позволяют использовать в качестве имени свойства любую строку:

|  |
| --- |
| var person = {};  person['любимый стиль музыки'] = 'Джаз'; |

# Домашнее задание

* Написать функцию преобразования цвета из 10-ного представления в 16-ный. Функция должна принимать 3 числа от 0 до 255 и возвращать строковый хеш.
* Написать функцию, преобразующую число в объект. Передавая на вход число от 0 до 999, мы должны получить на выходе объект, в котором в соответствующих свойствах описаны единицы, десятки и сотни. Например для числа 245 мы должны получить следующий объект: {‘единицы’: 5, ‘десятки’: 4, 'сотни’: 2}. Если число превышает 999, необходимо выдать соответствующее сообщение с помощью console.log и вернуть пустой объект.
* \* Реализовать функцию objectToQueryString(object), которая принимает аргументом объект, возвращает строку. Примеры работы:

|  |
| --- |
| console.log(objectToQueryString({user: 'Dmitry'})); *// user=Dmitry* console.log(objectToQueryString({user: 'Dmitry', password: 'pass'})); *// user=Dmitry&password=pass* console.log(objectToQueryString({user: 'Dmitry', password: 'pass', id=1})); *// user=Dmitry&password=pass&id=1* |

# Практика

Продолжаем адресную книгу:

1. Добавить всем пользователям id (поле, по которому пользователя можно уникально идентифицировать).

2. Переписать метод добавления таким образом, чтобы id генерировался автоматически (например, порядковый номер)

3. Добавить метод удаления пользователей

# Используемая литература

* [Mozilla Developer Network](https://developer.mozilla.org/)
* [MSDN](http://msdn.microsoft.com/)
* [Safari Developer Library](https://developer.apple.com/library/safari/navigation/index.html)
* [Современный учебник JavaScript](http://learn.javascript.ru)