



Структуры данных. Числа

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

```
alert(1 / 0); // Infinity  
alert(12345 / 0); // Infinity
```

```
alert(Infinity > 1234567890); // true  
alert(Infinity + 5 == Infinity); // true  
alert(-1 / 0); // -Infinity
```

NaN

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

```
alert(0 / 0); // NaN
```

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





parseInt и parseFloat

Функция parseInt и ее аналог parseFloat преобразуют строку символ за символом, пока это возможно.

```
alert( parseInt('12px') ) // 12, ошибка на символе 'p'  
alert( parseFloat('12.3.4') ) // 12.3, ошибка на второй точке
```

```
1 alert( parseInt('a123') ); // NaN
```

Функция parseInt читает из строки целое число, а parseFloat – дробное

```
alert( parseInt('12px') ) // 12, ошибка на символе 'p'  
alert( parseFloat('12.3.4') ) // 12.3, ошибка на второй точке
```





Округление

Одна из самых частых операций с числом – округление. В JavaScript существуют целых 3 функции для этого.

Math.floor

Округляет в меньшую сторону

Math.ceil

Округляет в большую сторону

Math.round

Округляет до ближайшего целого

```
alert( Math.floor(3.1) ); // 3  
alert( Math.ceil(3.1) ); // 4  
alert( Math.round(3.1) ); // 3
```





Округление

Существует также специальный метод `num.toFixed(precision)`, который округляет число `num` до точности `precision` и возвращает результат в виде строки:

```
var n = 12.34;  
alert( n.toFixed(1) ); // "12.3"
```

Округление идёт до ближайшего значения, аналогично `Math.round`
Итоговая строка, при необходимости, дополняется нулями до нужной точности:

```
var n = 12.34;  
alert( n.toFixed(5) ); // "12.34000", добавлены нули до 5 знаков после запятой
```





Создание строк

Строки создаются при помощи двойных или одинарных кавычек:

Округление идёт до ближайшего значения, аналогично `Math.round`
Итоговая строка, при необходимости, дополняется нулями до нужной точности:

```
1 var text = "моя строка";  
2  
3 var anotherText = 'еще строка';  
4  
5 var str = "012345";
```

В JavaScript нет разницы между двойными и одинарными кавычками.





Создание строк

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

```
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 ©
--------	---





Методы и свойства

Длина length

```
var str = "My\n"; // 3 символа. Третий - перевод строки  
alert( str.length ); // 3
```

Доступ к символам

Чтобы получить символ, используйте вызов `charAt(позиция)`. Первый символ имеет позицию 0:

```
var str = "jQuery";  
alert( str.charAt(0) ); // "j"
```

Также для доступа к символу можно также использовать квадратные скобки:

```
var str = "Я - современный браузер!";  
alert( str[0] ); // "Я"
```





Методы и свойства

Разница между этим способом и `charAt` заключается в том, что если символа нет – `charAt` выдает пустую строку, а скобки – `undefined`:

```
1 alert( "" .charAt(0) ); // пустая строка
2 alert( "" [0] ); // undefined
```

Смена регистра

Методы `toLowerCase()` и `toUpperCase()` меняют регистр строки на нижний/верхний:

```
alert( "Интерфейс".toUpperCase() ); // ИНТЕРФЕЙС
```

```
alert( "Интерфейс" [0].toLowerCase() ); // 'и'
```





Поиск подстроки

Для поиска подстроки есть метод `indexOf(подстрока[, начальная_позиция])`.

Он возвращает позицию, на которой находится подстрока или -1, если ничего не найдено. Например:

```
var str = "Widget with id";

alert( str.indexOf("Widget") ); // 0, т.к. "Widget" найден прямо в начале str
alert( str.indexOf("id") ); // 1, т.к. "id" найден, начиная с позиции 1
alert( str.indexOf("widget") ); // -1, не найдено, так как поиск учитывает регистр
```

Необязательный второй аргумент позволяет искать, начиная с указанной позиции. Например, первый раз "id" появляется на позиции 1. Чтобы найти его следующее появление – запустим поиск с позиции 2:

```
var str = "Widget with id";

alert(str.indexOf("id", 2)) // 12, поиск начат с позиции 2
```





Поиск всех вхождений

Чтобы найти все вхождения подстроки, нужно запустить `indexOf` в цикле. Как только получаем очередную позицию – начинаем следующий поиск со следующей.

```
var str = "Ослик Иа-Иа посмотрел на виадук"; // ищем в этой строке
var target = "Иа"; // цель поиска

var pos = 0;
while (true) {
    var foundPos = str.indexOf(target, pos);
    if (foundPos == -1) break;

    alert( foundPos ); // нашли на этой позиции
    pos = foundPos + 1; // продолжить поиск со следующей
}
```

Впрочем, тот же алгоритм можно записать и короче:

```
var str = "Ослик Иа-Иа посмотрел на виадук"; // ищем в этой строке
var target = "Иа"; // цель поиска

var pos = -1;
while ((pos = str.indexOf(target, pos + 1)) != -1) {
    alert( pos );
}
```





Взятие подстроки: substr, substring.

substring(start [, end])

Метод substring(start, end) возвращает подстроку с позиции start до, но не включая end. Если аргумент end отсутствует, то идет до конца строки

```
var str = "stringify";  
alert(str.substring(0,1)); // "s", символы с позиции 0 по 1 не включая 1.  
alert(str.substring(2)); // ringify, символы с позиции 2 до конца
```

substr(start [, length])

Первый аргумент имеет такой же смысл, как и в substring, а второй содержит не конечную позицию, а количество символов.

```
var str = "stringify";  
str = str.substr(2,4); // ring, со 2-й позиции 4 символа  
alert(str)
```





Взятие подстроки: slice.

`slice(start [, end])`

Возвращает часть строки от позиции `start` до, но не включая, позиции `end`.
Смысл параметров – такой же как в `substring`.

Различие между `substring` и `slice` – в том, как они работают с отрицательными и выходящими за границу строки аргументами

```
alert( "testme".substring(-2) ); // "testme", -2 становится 0
```

`substring(start, end)`

Отрицательные аргументы интерпретируются как равные нулю. Слишком большие значения усекаются до длины строки

```
alert( "testme".slice(-2) ); // "me", от 2 позиции с конца
```

```
alert( "testme".slice(1, -1) ); // "estm", от 1 позиции до первой с конца.
```





Операции с объектом

Ассоциативный массив – структура данных, в которой можно хранить любые данные в формате ключ-значение.

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
```





Операции с объектом

В JavaScript можно обратиться к любому свойству объекта, даже если его нет.

Ошибки не будет.

Но если свойство не существует, то вернется специальное значение `undefined`.

Таким образом мы можем легко проверить существование свойства –
полу

```
var person = {  
  name: "Василий"  
};  
  
alert( person.lalala === undefined ); // true, свойства нет  
alert( person.name === undefined ); // false, свойство есть.
```





Операции с объектом

Доступ через квадратные скобки

Существует альтернативный синтаксис работы со свойствами, использующий квадратные скобки объект['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';
```





Операции с объектом

Доступ к свойству через переменную

Квадратные скобки также позволяют обратиться к свойству, имя которого хранится в переменной:

```
var person = {};  
person.age = 25;  
var key = 'age';  
  
alert( person[key] ); // выведет person['age']
```

Вообще, если имя свойства хранится в переменной (`var key = "age"`), то единственный способ к нему обратиться – это квадратные скобки `person[key]`.

Доступ через точку используется, если мы на этапе написания программы уже знаем название свойства. А если оно будет определено по ходу выполнения, например, введено посетителем и записано в переменную, то единственный выбор – квадратные скобки.





Операции с объектом

Объявление со свойствами

Объект можно заполнить значениями при создании, указав их в фигурных скобках: { ключ1: значение1, ключ2: значение2, ... }.

Такой синтаксис называется *литеральным* (англ. literal).

```
var menuSetup = {  
  width: 300,  
  height: 200,  
  title: "Menu"  
};  
  
// то же самое, что:  
  
var menuSetup = {};  
menuSetup.width = 300;  
menuSetup.height = 200;  
menuSetup.title = 'Menu';
```





Операции с объектом

Названия свойств можно перечислять как в кавычках, так и без, если они удовлетворяют ограничениям для имён переменных.

```
var menuSetup = {  
  width: 300,  
  'height': 200,  
  "мама мыла раму": true  
};
```

В качестве значения можно тут же указать и другой объект:

```
var user = {  
  name: "Таня",  
  age: 25,  
  size: {  
    top: 90,  
    middle: 60,  
    bottom: 90  
  }  
}  
  
alert(user.name) // "Таня"  
  
alert(user.size.top) // 90
```





Объекты. Проверочное задание

Необходимо создать объект *учебник*, который будет содержать следующие данные (подставьте свои значения):

- Автор
- Год издания
- Количество страниц
- Издательство
- Предмет
- Класс
- Язык





Объекты: перебор свойств

Для перебора всех свойств из объекта используется цикл по свойствам `for..in`. Эта синтаксическая конструкция отличается от рассмотренного ранее цикла `for(;;)`.

```
for (key in obj) {  
    /* ... делать что-то с obj[key] ... */  
}
```

```
var menu = {  
    width: 300,  
    height: 200,  
    title: "Menu"  
};  
  
for (var key in menu) {  
    // этот код будет вызван для каждого свойства объекта  
    // ..и выведет имя свойства и его значение  
  
    alert( "Ключ: " + key + " значение: " + menu[key] );  
}
```

Последовательно
по свойствам объекта `obj`,
каждое свойство
будет выведено
на экран. Этот цикл
называется **циклом по свойствам**.





Объекты: количество свойств

Как узнать, сколько свойств хранит объект?

Готового метода для этого нет.

Самый кросс-браузерный способ – это сделать цикл по свойствам и посчитать, вот так:

```
var menu = {  
  width: 300,  
  height: 200,  
  title: "Menu"  
};  
  
var counter = 0;  
  
for (var key in menu) {  
  counter++;  
}  
  
alert( "Всего свойств: " + counter );
```





Копирование по ссылке

В переменной, которой присвоен объект, хранится не сам объект, а «адрес его места в памяти», иными словами – «ссылка» на него.

При копировании переменной с объектом – копируется эта ссылка, а объект по-прежнему остается в единственном экземпляре.

```
var user = { name: 'Вася' };  
  
var admin = user;  
  
admin.name = 'Петя'; // поменяли данные через admin  
alert(user.name); // 'Петя', изменения видны в user
```





Клонирование объектов

В нижепоказанном коде каждое свойство объекта user копируется в clone. Если предположить, что они примитивны, то каждое скопируется по значению и мы как раз получим полный клон.

```
var user = {  
  name: "Вася",  
  age: 30  
};  
  
var clone = {}; // новый пустой объект  
  
// скопируем в него все свойства user  
for (var key in user) {  
  clone[key] = user[key];  
}  
  
// теперь clone - полностью независимая копия  
clone.name = "Петя"; // поменяли данные в clone  
  
alert( user.name ); // по-прежнему "Вася"
```





Массивы с числовыми индексами

Массив – разновидность объекта, которая предназначена для хранения пронумерованных значений и предлагает дополнительные методы для удобного манипулирования такой коллекцией. Они обычно используются для хранения упорядоченных коллекций данных, например – списка товаров на странице, студентов в группе и т.п.

Элементы нумеруются, начиная с нуля.

Чтобы получить нужный элемент из массива – указывается его номер в квадратных скобках:

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





Массивы с числовыми индексами

Элемент можно всегда заменить или добавить:

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

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

Через alert можно вывести и массив целиком.

При этом его элементы будут перечислены через запятую:

```
1 var fruits = ["Яблоко", "Апельсин", "Груша"];  
2  
3 alert( fruits ); // Яблоко,Апельсин,Груша
```





Массивы с числовыми индексами

В массиве может храниться любое число элементов любого типа.:

```
// микс значений  
var arr = [ 1, 'Имя', { name: 'Петя' }, true ];  
  
// получить объект из массива и тут же -- его свойство  
alert( arr[2].name ); // Петя
```





Методы pop/push

pop

Удаляет последний элемент из массива и возвращает его:

```
var fruits = ["Яблоко", "Апельсин", "Груша"];  
alert( fruits.pop() ); // удалили "Груша"  
alert( fruits ); // Яблоко, Апельсин
```

push

Добавляет элемент в конец массива:

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





Методы shift/unshift

shift

Удаляет из массива первый элемент и возвращает его:

```
var fruits = ["Яблоко", "Апельсин", "Груша"];  
alert( fruits.shift() ); // удалили Яблоко  
alert( fruits ); // Апельсин, Груша
```

unshift

Добавляет элемент *в начало* массива:

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





Перебор элементов

Для перебора элементов обычно используется цикл:

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





Особенности работы length

Встроенные методы для работы с массивом автоматически обновляют его длину length.

Длина length – не количество элементов массива, а последний индекс + 1.

Так уж оно устроено.

Это легко увидеть на следующем примере:

```
1 var arr = [];  
2 arr[1000] = true;  
3  
4 alert(arr.length); // 1001
```





Массивы Метод split

Ситуация из реальной жизни. Мы пишем сервис отсылки сообщений и посетитель вводит имена тех, кому его отправить: Маша, Петя, Марина, Василий.... Но нам-то гораздо удобнее работать с массивом имен, чем с одной строкой.

К счастью, есть метод `split(s)`, который позволяет превратить строку в массив, разбив ее по разделителю `s`. В примере ниже таким разделителем является строка из запятой и пробела.

```
var names = 'Маша, Петя, Марина, Василий';  
  
var arr = names.split(', ');  
  
for (var i = 0; i < arr.length; i++) {  
    alert( 'Вам сообщение ' + arr[i] );  
}
```





Массивы Метод join

Вызов `arr.join(str)` делает в точности противоположное `split`. Он берет массив и склеивает его в строку, используя `str` как разделитель.

```
var arr = ['Маша', 'Петя', 'Марина', 'Василий'];  
var str = arr.join(';');  
alert( str ); // Маша;Петя;Марина;Василий
```





Удаление из массива

Так как массивы являются объектами, то для удаления ключа можно воспользоваться обычным delete:

```
var arr = ["Я", "иду", "домой"];  
  
delete arr[1]; // значение с индексом 1 удалено  
  
// теперь arr = ["Я", undefined, "домой"];  
alert( arr[1] ); // undefined
```





Метод splice

Метод `splice` – это универсальный раскладной нож для работы с массивами. Умеет все: удалять элементы, вставлять элементы, заменять элементы – по очереди и одновременно.

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

Удалить `deleteCount` элементов, начиная с номера `index`, а затем вставить `elem1, ..., elemN` на их место. Возвращает массив из удалённых элементов. Этот метод проще всего понять, рассмотрев примеры.

```
var arr = ["Я", "изучаю", "JavaScript"];  
  
arr.splice(1, 1); // начиная с позиции 1, удалить 1 элемент  
  
alert( arr ); // осталось ["Я", "JavaScript"]
```





Метод slice

Метод slice(begin, end) копирует участок массива от begin до end, не включая end. Исходный массив при этом не меняется.

```
var arr = ["Почему", "надо", "учить", "JavaScript"];  
var arr2 = arr.slice(1, 3); // элементы 1, 2 (не включая 3)  
alert( arr2 ); // надо, учить
```

