|  |  |  |
| --- | --- | --- |
| № | Код | Объяснение |
| 1 | arr = "a,b,c".split(',') // массив ["a", "b", "c"]  var tags = "Лето, зима, осень 2009 года"  var re = /\s\*,\s\*/  var tagList = tags.split(re);  alert(tagList) // ["Лето", "зима", "осень 2009 года"]  var str ="a b с d"  str.split(' ',2)  // ["a", "b"], а не ["a" "b c d"]  var myString = "Hello 1 word. Sentence number 2."  var splits = myString.split(/(\d)/)  // splits = ["Hello ", "1", " word. Sentence number ", "2", "."] | Метод split возвращает новый массив.  Строка бьется по separator, при разбивании separator пропадает  Если separator не указан или совпадений нет, то возвращаемый массив состоит из одного элемента - исходной строки.  Если separator является регулярным выражением со вложенными скобками, то каждый раз при совпадении separator все результаты захвата скобками, включая undefined результаты, добавляются в выходной массив.  Если вызывающая строка пуста, то split возвращает массив из одной пустой строки, а не пустой массив.  **Аргументы**  **separator**  регулярное выражение или строка, по которой делить str  **limit**  максимальное количество кусков, на которые может быть разбита строка  Следующий пример делит по ',' и убирает лишние пробелы вокруг запятой.  При делении с ограничением limit - элементы массива свыше limit будут отброшены.  Пример: Ограничение массива  Следующий пример иллюстрирует добавление в массив результатов захвата вложенными скобками.  Благодаря скобкам, найденный разделитель separator сохранился и вошел в результирующий массив. |
| 2 | function fixedCharCodeAt (str, idx) {  var code = str.charCodeAt(idx);  if (0xD800 <= code && code <= 0xDBFF) {  // Верхний вспомогательный символ  var hi = code;  var low = str.charCodeAt(idx+1);  return ((hi - 0xD800) \* 0x400) + (low - 0xDC00) + 0x10000;  }  if (0xDC00 <= code && code <= 0xDFFF) {  // Нижний вспомогательный символ  var hi = str.charCodeAt(idx-1);  var low = code;  return ((hi - 0xD800) \* 0x400) + (low - 0xDC00) + 0x10000;  }  return code;  }  alert(fixedCharCodeAt ('\uD800\uDC00', 0)); // 65536  alert(fixedCharCodeAt ('\uD800\uDC00', 1)); // 65536  // возвратит 65, unicode-код для А  "ABC".charCodeAt(0) // 65  var str = 'A\uD800\uDC00Z' // 2 символа  // мы могли объявить сложный символ напрямую,  // т.к наш javascript-код имеет кодировку UTF.  for (var i=0, chr; i < str.length; i++) {  if ((chr = getWholeChar(str, i)) === false) {  continue // на позиции i вспомогательный символ  }  // alert будет последовательно вызван для всех символов  alert(chr);  }  function getWholeChar (str, i) {  var code = str.charCodeAt(i);  if (0xD800 <= code && code <= 0xDBFF) {  // верхний вспомогательный  if (str.length <= (i+1)) {  throw 'High surrogate without following low surrogate';  }  var next = str.charCodeAt(i+1);  if (0xDC00 > next || next > 0xDFFF) {  throw 'High surrogate without following low surrogate';  }  return str[i]+str[i+1];  }  else if (0xDC00 <= code && code <= 0xDFFF) {  // Нижний вспомогательный  if (i === 0) {  throw 'Low surrogate without preceding high surrogate';  }  var prev = str.charCodeAt(i-1);  if (0xD800 > prev || prev > 0xDBFF) {  throw 'Low surrogate without preceding high surrogate';  }  return false;  }  return str[i] // используем обращение к строке как к массиву  } | Юникодное значение от 0 до 1,114,111. Первые 128 значений Unicode совпадают с кодировкой ASCII.  Заметим, что charCodeAt() всегда возвращает значение, меньшее 65536. Это - из за того, что более высокие юникод-символы представлены парой вспомогательных псевдо-символов, которые вместе составляют реальный символ.  Из-за этого, чтобы получить полный символ для символов, юникод-значение которых больше или равно 65536, необходимо получить не только charCodeAt(0), но и charCodeAt(1) (как для строки из двух букв).  charCodeAt() возвращает NaN, если указанный индекс меньше нуля или больше/равен длине строки.  Следующий пример предоставляет способ убедиться, что проход по циклу всегда дает целый символ, даже если строка содержит символы за пределами основного многоязычного набора. |
| 3 | function fixedFromCharCode (codePt) {  if (codePt > 0xFFFF) {  codePt -= 0x10000;  return String.fromCharCode(0xD800 + (codePt >> 10), 0xDC00 + (codePt & 0x3FF));  } else {  return String.fromCharCode(codePt);  }  } | **Синтаксис**   |  | | --- | | String.fromCharCode(num1, ..., numN) |   **Аргументы**  **num1, ..., numN**  последовательность чисел, представляющих собой коды Unicode-символов  **Описание, примеры**  Этот метод возвращает элементарную строку, а не объект типа String.  Так как fromCharCode - статический метод String, он всегда вызывается как String.fromCharCode().  Хотя наиболее частые значения в юникоде могут быть представлены одним числом (как задумано в ранних стандартах яваскрипт), и fromCharCode() можно использовать, чтобы получить один символ для частых значений (т.е значений UCS-2, которые являются подмножеством UTF-16), но для ВСЕХ юникодных значений одного fromCharCode() не хватает.  Так как более редкие символы кодируются в юникоде с использованием пары чисел, fromCharCode() можно использовать для возврата этой пары, и, таким образом, получения нужного значения.  Однако, имейте в виду, что следующая вспомогательная функция, которая получает любой юникодный символ, может вернуть такое значение, которое показывается как один символ, но в строке занимает 2 места. Хотя обычно - это все же будет 1 символ и 1 позиция в строке. |
| 4 | var anyString="Привет, мир"  document.writeln("Символ на позиции 0 '" + anyString.charAt(0) + "'")  document.writeln("Символ на позиции 1 '" + anyString.charAt(1) + "'")  document.writeln("Символ на позиции 2 '" + anyString.charAt(2) + "'")  document.writeln("Символ на позиции 3 '" + anyString.charAt(3) + "'")  document.writeln("Символ на позиции 4 '" + anyString.charAt(4) + "'")  document.writeln("Символ на позиции 999 '" + anyString.charAt(999) + "'")  Символ на позиции 0 'П'  Символ на позиции 1 'р'  Символ на позиции 2 'и'  Символ на позиции 3 'в'  Символ на позиции 4 'е'  Символ на позиции 999 '' | **charAt**  **Синтаксис**   |  | | --- | | str.charAt(index) |   **Аргументы**  **index**  Целое число от 0 до количества символов в строке минус 1  **Описание, примеры**  Символы идут слева направо. Первый символ имеет позицию 0, а последний string.length-1.  Если указанный index лежит за пределами этого диапазона, яваскрипт вернет пустую строку. |
| 5 | a = "строка"  b = " очень"  с = " длинная"  alert(a.concat(b,c)) // строка очень длинная | **concat**  **Синтаксис**   |  | | --- | | result = string1.concat(string2, string3[, ..., stringN]) |   **Аргументы**  **string2...stringN**  сроки, которые будут прибавлены к string1  **Описание, примеры**  Выполняет действие, аналогичное оператору + для строк. |
| 6 | "canal".lastIndexOf("a") // 3  "canal".lastIndexOf("a",2) // 1  "canal".lastIndexOf("a",0) // -1  "canal".lastIndexOf("x") // -1  "Привет".lastIndexOf("п") // -1 | **lastIndexOf**  **Синтаксис**   |  | | --- | | str.lastIndexOf(searchValue[, fromIndex]) |   **Аргументы**  **searchValue**  искомая подстрока  **fromIndex**  индекс, от которого начинать поиск по направлению к началу строки. От 0 до str.length-1. Значение по умолчанию: str.length  **Описание, примеры**  Символы в строке идут слева направо. Позиция первого символа 0, последнего - длина строки минус один.  Метод lastIndexOf чувствителен к регистру символов. |
| 7 | function testinput(re, str){  if (str.search(re) != -1)  return true  else  return false  }  alert(testinput(/something/, "somestr")) // false | **search**  **Синтаксис**   |  | | --- | | idx = str.search(regexp) |   **Аргументы**  **regexp**  объект типа [RegExp](http://javascript.ru/RegExp) или строка  **Описание, примеры**  Этот метод удобен, когда нужно проверить, есть ли совпадения с регулярным выражением (аналогично [RegExp#test](http://javascript.ru/RegExp/test)).  Чтобы получить больше информации, используйте более медленный метод [match](http://javascript.ru/String/match) (или родственный ему метод [RegExp#exec](http://javascript.ru/RegExp/exec)). |
| 8 | str = "За информацией обратитесь: Глава 3.4.5.1";  re = /глава (\d+(\.\d)\*)/i  found = str.match(re)  document.write(found)  str = "За информацией обратитесь: Глава 3.4.5.1, Глава 7.5";  re = /глава (\d+(\.\d)\*)/ig  found = str.match(re)  alert(found) | **match**  **Синтаксис**   |  | | --- | | str.match(regexp) |   **Аргументы**  **regexp**  регулярное выражение в виде объекта [RegExp](http://javascript.ru/RegExp) или строка  **Описание, примеры**  Если регулярное выражение без флага "g", то возвращает такое же результат, как [regexp.exec(str)](http://javascript.ru/RegExp/exec).  Если же для регулярного выражения указан флаг "g", то метод возвращает массив, содержащий все совпадения.  Если совпадений нет, то метод возвращает не пустой массив, а null.  **Замечания**   * Если вы хотите только проверить, совпадает ли строка с регулярным выражением - используйсте [regexp.test(string)](http://javascript.ru/RegExp/test). * Если вам нужно только первое совпадение - вам может лучше подойти [regexp.exec(string)](http://javascript.ru/RegExp/exec)   **Пример без глобального поиска**  В следующем примере match используется для поиска строки "Глава", за которой идет 1 или групп из цифр с последующей точкой.  Возвратит массив из трех элементов:   |  | | --- | | ["Глава 3.4.5.1", "3.4.5.1", ".1"] |  1. "Глава 3.4.5.1" - полное совпадение с регулярным выражением /глава (\d+(\.\d)\*)/i, 2. "3.4.5.1" - первая скобка в совпадении, 3. ".1" - вторая скобка в совпадении   **Пример с глобальным поиском**  При глобальном поиске регультат match - просто массив всех совпадений (и null, если их нет).  Выведет массив из двух элементов:   |  | | --- | | [ "Глава 3.4.5.1", "Глава 7.5" ] |   Скобки при таком поиске не учитываются.  Если вам нужен глобальный поиск с учетом скобок - используйте многократный вызов [exec](http://javascript.ru/RegExp/exec). |
| 9 | var upperText="ПриВет"  document.write(upperText.toLowerCase()) // привет | **toLowerCase**  **Синтаксис**   |  | | --- | | str.toLowerCase() |   **Описание, примеры** |
| 10 | var upperText="ПриВет"  document.write(upperText.toUpperCase()) // ПРИВЕТ | **toUpperCase**  **Синтаксис**   |  | | --- | | str.toUpperCase() |   **Описание, примеры** |
| 11 |  | **toLocaleLowerCase**  **Синтаксис**   |  | | --- | | str.toLocaleLowerCase() |   **Описание, примеры**  Обычно работает в точности как [toLowerCase](http://javascript.ru/String/toLowerCase). Различия есть только для некоторых языков, например, для турецкого, правила которых противоречат соглашениям Unicode.  Для русского результаты совпадают. |
| 12 |  | **toLocaleUpperCase**  **Синтаксис**   |  | | --- | | str.toLocaleUpperCase() |   **Описание, примеры**  Обычно работает в точности как [toUpperCase](http://javascript.ru/String/toUpperCase). Различия есть только для некоторых языков, например, для турецкого, правила которых противоречат соглашениям Unicode.  Для русского результаты совпадают. |
| 13 | Нет упоминания о возможности изменения основания:  n.toString(16);  (215).toString(16); // "d7"  так например можно получать hex коды клавиш. | **toString**  **Синтаксис**   |  | | --- | | str.toString() |   **Описание, примеры**  Перекрывает соответствующий метод [Object#toString](http://javascript.ru/Object/toString), возвращает элементарную строку. |
| 14 | x = new String("Hello world");  alert(x.valueOf()) // покажет"Hello world" | **valueOf**  **Синтаксис**   |  | | --- | | str.valueOf() |   **Описание, примеры**  Метод valueOf возвращает элементарное значение для объекта [String](http://javascript.ru/String). Это значение эквивалентно [String#toString](http://javascript.ru/String/toString).  Обычно этот метод вызывается неявно, внутри интерпретатора javascript, а не в коде. |
| 15 | var str = "Моя строка"  str.substring(0,3) // Моя  str.substring(3,0) // Моя  str.substring(1,2) // о  str.substring(4) // "строка" | **substring**  **Синтаксис**   |  | | --- | | str.substring(indexA, [indexB]) |   **Аргументы**  **indexA**  целое число от 0 до длины строки-1  **indexB**  целое число от 0 до длины строки-1  **Описание, примеры**  Метод substring возвращает подстроку, начиная с позиции indexA до, но не включая indexB.  В частности:   * Если indexA = indexB, возвращается пустая строка * Если indexB не указан, substring возвращает символы до конца строки * Если какой-то из аргументов меньше 0 или является NaN - он считается равным 0 * Если какой-то из аргументов больше, чем длина строки - он считается равным длине строки   Если indexA > indexB, тогда substring ведет себя, как будто аргументы поменялись местами.  Например, str.substring(1, 0) == str.substring(0, 1). |
| 16 | var str ="Вот такая строка"  str.slice(0,2) // "Во"  str.slice(1,-1) // "от такая строк" | **slice**  **Синтаксис**   |  | | --- | | var sub = string.slice(beginSlice[, endSlice]) |   **Аргументы**  **beginSlice**  позиция, начиная от 0, с которой начинается участок  **endSlice**  позиция, на которой заканчивается участок  **Описание, примеры**  Метод slice возвращает часть строки от beginSlice до endSlice, но не включая символ под номером endSlice, не меняя вызывающую строку.  Например, string.slice(1,4) возвратит участок строку со 2го по 4й символы, т.е из символов с позицией 1, 2 и 3.  Если параметр endSlice меньше нуля, то он обозначает сдвиг относительно конца строки.  Например, string.slice(2,-1) вернет участок строки, начиная от 3го символа (с индексом 2), и кончая предпоследним. |
| 17 | "Привет, мир".indexOf("Привет") // вернет 0  "Привет, мир".indexOf("Корова") // вернет -1  "Привет, мир".indexOf("мир") // вернет 8  "Привет, мир".indexOf("Мир") // вернет -1  "Привет, мир".indexOf("Привет", 0) // вернет 0  "Привет, мир".indexOf("р", 1) // вернет 1  "Привет, мир".indexOf("р", 5) // вернет 10  "Привет, мир".indexOf("", 5) // вернет 5  "Привет, мир".indexOf("", 7) // вернет 7  "Привет, мир".indexOf("", 999) // вернет 11 (длину)  count = 0;  pos = str.indexOf("x");  while ( pos != -1 ) {  count++;  pos = str.indexOf("x",pos+1);  } | **indexOf**  **Синтаксис**   |  | | --- | | str.indexOf(searchValue[, fromIndex]) |   **Аргументы**  **searchValue**  искомая подстрока  **fromIndex**  позиция, с которой начинать поиск: от 0 до длины строки-1  **Описание, примеры**  Поиск осуществляется слева направо. Метод чувствителен к регистру символов.  **Пример: с позицией**  Пустая подстрока находится в любом случае:  **Пример: для пустой подстроки**  Следующий пример демонстрирует использование indexOf для подсчета количества повторений буквы в строке. |
| 18 | var str = "abcdefghij";  alert("(1,2): " + str.substr(1,2));  alert("(-3,2): " + str.substr(-3,2));  alert("(-3): " + str.substr(-3));  alert("(1): " + str.substr(1));  alert("(-20, 2): " + str.substr(-20,2));  alert("(20, 2): " + str.substr(20,2));  (1,2): bc  (-3,2): hi  (-3): hij  (1): bcdefghij  (-20, 2): ab  (20, 2): | **substr**  **Синтаксис**   |  | | --- | | var sub = string.substr(start[, length]) |   **Аргументы**  **start**  начальная позиция, начиная от 0  **length**  сколько символов брать с позиции start  **Описание, примеры**  Этот метод описан только в ненормативной части стандарта ECMA, но реализован во всех браузерах.  Метод substr начинает собирать строку с позиции start и заканчивает, когда соберет length или дойдет до конца строки.  Если start больше или равен длине строки, то substr возвращает пустую строку.  Если start < 0, позиция определяется с конца строки. При этом если полученная позиция - до начала строки, то в качестве start берется 0. Это поведение не поддерживается Internet Explorer.  Если параметр length не указан - подстрока берется до конца строки.  Если length <= 0 - возвращается пустая строка.  Последовательно выведет: |
| 19 | function replacer(str, p1, p2, offset, s) {  return p1 + ", " + p2;  }  newString = "my XXzz".replace(/(X+)(z+)/, replacer)   |  | | --- | | str = "тест еще тест" |  |  | | --- | | str.replace("тест","прошел") // = "прошел еще тест" |   str = "тест еще тест"  str.replace(/тест/g,"прошел") // = "прошел еще прошел"  // или так  str.replace(new RegExp("тест",'g'),"прошел")  var re = /(\w+)\s(\w+)/;  var str = "John Smith";  var newstr = str.replace(re, "$2, $1") // "Smith, John" | **replace**  **Синтаксис**   |  | | --- | | str.replace(regexp, newSubStr|function) |   **Аргументы**  **regexp**  объект типа [RegExp](http://javascript.ru/RegExp) или строка  **newSubStr|function**  новая подстрока или функция, которая ее возвращает  **Описание, примеры**  Этот метод не меняет вызывающую строку, а возвращает новую, после замен.  Чтобы произвести глобальный поиск и замену, используйте regexp c флагом "g".  **Строка замены**  Если вы указываете строку как второй параметр - она может включать в себя следующие специальные сочетания:**$$** Вставляет "$"  **$&** Вставляет совпавшую подстроку  **$`** Вставляет часть строки, которая предшествует совпавшей подстроке  **$'** Вставляет часть строки, которая следует за совпавшей подстрокой  **$n или $nn, где n/nn - десятичные цифры**  Вставляет n-ю скобку в совпадении, если первый аргумент - объект RegExp  **Функция замены**  Если вы указываете функцию в качестве второго параметра, то она вызывается при каждом совпадении.  Ваша функция может динамически генерировать строку, которая заменит совпавшую подстроку. В качестве значения для замены используется результат работы функции.  Функция может использовать совпавшие подстроки - она их получает в параметрах вызова. Первый параметр всегда содержит полную совпавшую подстроку.  Если при вызове replace указано регулярное выражение, а не строка, то последующие параметры будут содержать значения скобочных групп.  Наконец, последние два параметра - позиция, на которой произошло совпадения и вызывающая строка.  Следующий пример вернет my XX, zz.  Значения параметров при вызове replacer:  **Str** "XXzz" - совпавшая подстрока  **p1** "XX" - первая скобка  **p2** "zz" - вторая скобка  **offset** 3 - позиция в тексте для поиска  **s** "my XXzz" - вызывающая строка, т.е строка для поиска  В следующем примере используется строка замены. Глобального поиска при этом не происходит.  **Пример: строка замены**  Для глобального поиска используйте регулярное выражение.  **Пример: Глобальная замена строки**  Следующий пример использует при замене сочетания $1 и $2 для ссылок на результат поиска.  **Пример: Замена со ссылками** |
|  | var x = "Тест"  alert("Длина равна " + x.length) // 4 | **length**  **Длина строки**  **Описание, примеры**  **Свойство содержит количество юникодных символов в строке. Для пустой строки, length равно 0.** |

<http://javascript.ru/string>