Сегодня мы поговорим о регулярных выражениях. Что же такое регулярные выражения? Регулярные выражения - это шаблоны используемые для сопоставления последовательностей символов в строках.

Синтаксис создания регулярного выражения:

var regexp = /шаблон/; // без флагов

var regexp = /шаблон/gmi; // с флагами gmi (рассмотрим их дальше).

Шаблон или паттерн - основа регулярного выражения. Это строка, которую можно расширить специальными символами, которые делают поиск намного гибче.

В простейшем случае, если флагов и специальных символов нет, поиск по паттерну – то же самое, что и обычный поиск подстроки:

[**Флаги**](https://learn.javascript.ru/regexp-introduction#%D1%84%D0%BB%D0%B0%D0%B3%D0%B8)

Регулярные выражения могут иметь флаги, которые влияют на поиск. Они являются необязательными, по умолчанию не установлен ни один, но их использование позволяет сделать поиск намного гибче и мощнее.

В JavaScript их всего три:

**i -** если этот флаг есть, то регэксп ищет независимо от регистра.

Например:  
 var str = "БОЛЬШЕ регулярных выражений";

console.log(str.search(/больше/));//вернет -1

console.log(str.search(/больше/i));// вернет 0

**g -** если этот флаг есть, то регэксп ищет все совпадения, иначе – только первое.

console.log('все, любят, читать'.replace(/,/g, ''))// найдет и уберет все запятые в тексте

**m -** Многострочный режим.

Пример применения флага m.

var str = '1е место: Винни\n' +

'2е место: Пятачок\n' +

'33е место: Слонопотам';

alert( str.match(/^\d+/gm) ); // 1, 2, 33

Далее, мы рассмотрим методы RegExp. Методы для поиска по регулярным выражениям встроены прямо в обычные строки String. Существуют следующие встроенные методы работы с регулярными выражениями.

## str.search(reg): возвращает позицию первого совпадения или -1, если ничего не найдено. Ищет только первое совпадение.

## Пример использования данного метода мы видели выше.

## **str.match(reg):** метод str.match работает по-разному, в зависимости от наличия или отсутствия флага g. Cначала мы разберём вариант, когда его нет.

В этом случае str.match(reg) находит только одно, первое совпадение.

Результат вызова – это массив, состоящий из этого совпадения, с дополнительными свойствами index – позиция, на которой оно обнаружено и input – строка, в которой был поиск.

Пример**:**

var str = "some string";

var reg = str.match(/som/);

alert(reg[0])// some (совпадение в строке);

alert(reg.index)// 0 (позиция);

alert(reg.input) //"some new string";

Далее, рассмотрим вариант str.match(reg) с флагом g. При наличии флага g, вызов match возвращает обычный массив из всех совпадений.

Никаких дополнительных свойств у массива в этом случае нет, скобки дополнительных элементов не порождают.

Пример**:**

var str = "some new string with something interesting";

var reg = str.match(/som/g);

alert(reg)// вернет массив ["some"," some"]

## str.split(reg|substr, limit): разбивает строку в массив по разделителю – регулярному выражению regexp или подстроке substr.

Пример**:**

alert('12-34-56'.split(/-/)) // [12, 34, 56]

## str.replace(reg, str|func)

Многофункциональный инструмент для работы со строками, поиска и замены любого уровня сложности.

Пример использования этого метода мы уже видели.

console.log('все, любят, читать'.replace(/,/g, ''))// 'все любят читать'

В случае, если нам нужен более гибко настраиваемый вариант замены, мы использовать в качестве второго аргумента функцию. Она будет вызвана для каждого совпадения, и её результат будет вставлен как замена.

Например:

var i = 0;

// заменить каждое вхождение "ой" на результат вызова функции

alert("ОЙ-Ой-ой".replace(/ой/gi, function() {

return ++i;

})); // 1-2-3

Эта функция получает следующие аргументы:

str – найденное совпадение,

p1, p2, ..., pn – содержимое скобок (если есть),

offset – позиция, на которой найдено совпадение,

s – исходная строка.

## regexp.test(str): метод test проверяет, есть ли хоть одно совпадение в строке str. Возвращает true/false.

Работает, по сути, так же, как и проверка str.search(reg) != -1, например:

var str = "some string";

var regexp = /some/;

alert(regexp.test(str))//вернет true

**Классы символов**

Класс символов – это специальное обозначение, под которое подходит любой символ из определённого набора.

Наиболее часто используются:

**\d -** цифра, символ от 0 до 9.

**\s  -** пробельный символ, включая табы, переводы строки и т.п.

**\w -** символ «слова», а точнее – буква латинского алфавита или цифра или подчёркивание '\_'. Не-английские буквы не являются \w, то есть русская буква не подходит.

Пример**:**

var str = " Будущее за HTML6";

var reg = / HTML\d/

alert( str.match(reg) )// HTML6

еще пример**:**

alert( "Я люблю HTML5!".match(/\s\w\w\w\w\d/) ); // 'HTML5'

## [Обратные классы](https://learn.javascript.ru/regexp-character-classes#%D0%BE%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D0%B5-%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D1%8B)

Для каждого класса существует «обратный ему», представленный такой же, но заглавной буквой.

«Обратный» – означает, что ему соответствуют все остальные символы, например:

**\D -** не-цифра, то есть любой символ кроме \d, например буква.

**\S -** не-пробел, то есть любой символ кроме \s, например буква.

**\W -** любой символ, кроме \w, то есть не латиница, не подчёркивание, не цифра. В частности, русские буквы принадлежат этому классу.

Пример**:**

var str = "some, thing";

alert( str.replace(/\W/g, "") ); // something

# Наборы и диапазоны [...]

Если в регулярном выражении несколько символов или символьных классов заключены в квадратные скобки […], то это означает "искать любой символ из указанных в […]".

Например, [a-z] – произвольный символ от a до z, [0-5] – цифра от 0 до 5.

В примере ниже мы будем искать "x", после которого идёт два раза любая цифра или буква от A до F:

// найдёт "xAF"

alert( "Exception 0xAF".match(/x[0-9A-F][0-9A-F]/g) );

**Диапазоны «кроме»** - кроме обычных, существуют также исключающие диапазоны: [^…].

Квадратные скобки, начинающиеся со знака каретки: [^…] находят любой символ, кроме указанных.

alert( "alice15@gmail.com".match(/[^\d\sA-Z]/gi) ); // "@", "."

Сегодня мы рассмотрели что такое регулярные выражения и способы их применения. Надеюсь, мне удалось убедить Вас в том, что регулярные выражения - это достаточно простой и интересный инструмент, который может значительно упростить жизнь любому разработчику.