# Регулярные выражения

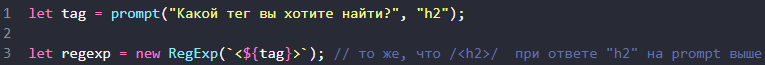
## Введение: шаблоны и флаги

Регулярки реализованы отдельным объектом RegExp и интегрированы в методы строк.  
Регулярка состоит из шаблона (также говорят «паттерн») и необязательных флагов.

* Длинный синтаксис:



Этот синтаксис допускает вставку переменных через ${...}:



* Короткий синтаксис:



Слэши играют здесь ту же роль, что и кавычки для обозначения строк.

### Флаги

* **i** - С этим флагом поиск не зависит от регистра: нет разницы между A и a (см. пример ниже).
* **g** - С этим флагом поиск ищет все совпадения, без него – только первое.
* **m** - Многострочный режим (рассматривается в главе Многострочный режим якорей ^ $, флаг "m").
* **s** - Включает режим «dotall», при котором точка . может соответствовать символу перевода строки \n (рассматривается в главе Символьные классы).
* **u** - Включает полную поддержку Юникода. Флаг разрешает корректную обработку суррогатных пар (подробнее об этом в главе Юникод: флаг "u" и класс \p{...}).
* **y** - Режим поиска на конкретной позиции в тексте (описан в главе Поиск на заданной позиции, флаг "y")

### Поиск: str.match(regexp)

Для строки str возвращает совпадения с регулярным выражением regexp

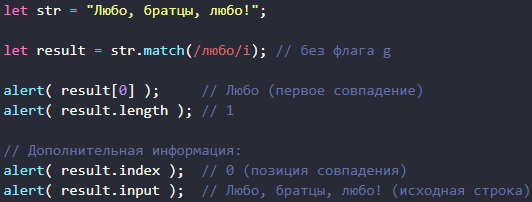
Три режима работы:

1. Если есть флаг g, то он возвращает массив всех совпадений:



(флаг i – для регистронезависимого поиска)

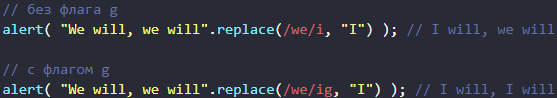
1. Если без g, то возвращает первое совпадение в виде массива, в котором по индексу 0 находится совпадение, и есть свойства с дополнительной информацией о нём:

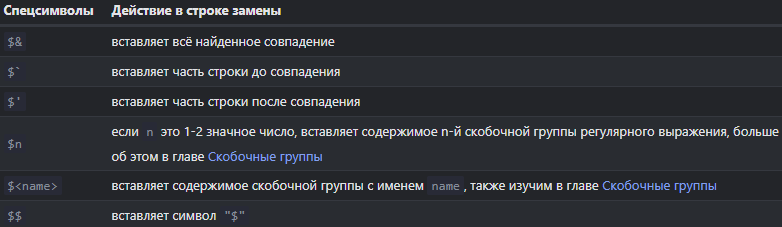
  
В этом массиве могут быть и другие индексы, кроме 0, если часть регулярного выражения выделена в скобки. Мы разберём это в главе Скобочные группы.

1. если совпадений нет, то, вне зависимости от наличия флага g, возвращается null.

### Замена: str.replace

Метод str.replace(regexp, replacement) заменяет совпадения с regexp в строке str на replacement (все, если есть флаг g, иначе только первое).



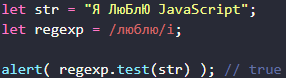
В строке замены replacement можно использовать спец. символы

Пример с $&:



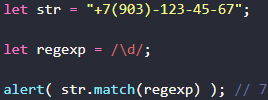
### Проверка: regexp.test

Если есть хоть одно совпадение возвращает true, иначе false.



## Символьные классы

Символьный класс – это специальное обозначение, которое соответствует любому символу из определённого набора.

Рассмотрим класс «цифра». Он обозначается как **\d** и в регулярном выражении соответствует «любой одной цифре».  


Если добавить флаг **g**, то найдет все цифры.

Наиболее используемые классы:

* **\d** («d» от английского «digit» означает «цифра»)

Цифра: символ от 0 до 9.

* **\s** («s»: от английского «space» – «пробел»)

Пробельные символы: включает в себя символ пробела, табуляции \t, перевода строки \n и некоторые другие редкие пробельные символы, обозначаемые как \v, \f и \r.

* **\w** («w»: от английского «word» – «слово»)

Символ «слова», а точнее – буква латинского алфавита или цифра или подчёркивание \_. Нелатинские буквы не являются частью класса \w, то есть буква русского алфавита не подходит.

Для примера, *\d\s\w* обозначает «цифру», за которой идёт пробельный символ, а затем символ слова, например “1 a”.

**Регулярное выражение может содержать как обычные символы, так и символьные классы.**

Например, *CSS\d* соответствует строке “CSS” с цифрой после неё:



Также мы можем использовать несколько символьных классов:



### Обратные символьные классы

Для каждого символьного класса существует «обратный класс», обозначаемый той же буквой, но в верхнем регистре.

* **\D**

Не цифра: любой символ, кроме \d, например буква.

* **\S**

Не пробел: любой символ, кроме \s, например буква.

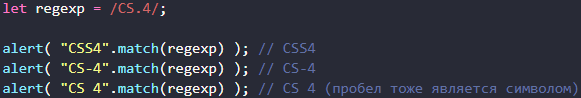
* **\W**

Любой символ, кроме \w, то есть не буквы из латиницы, не знак подчёркивания и не цифра. В частности, русские буквы принадлежат этому классу.

Альтернативный, более короткий путь – найти нецифровые символы \D и удалить их из строки:



### Точка – это любой символ



**Точка как буквально любой символ, с флагом «s»**

Обычно точка не соответствует символу новой строки \n. То есть, регулярное выражение A.B будет искать символ A и затем B, с любым символом между ними, кроме перевода строки \n:



Если нужно обозначить действительно «любой символ», включая перевод строки, то нужно использлвать флаг **s**:



## Юникод: флаг "u" и класс \p{…}

В js для строк используется кодировка Юникод. Символы кодируются с помощью 2 байтов, что позволяет закодировать максимум 65536 символов. Этого не хватает для того, чтобы закодировать все символы. Поэтому некоторые редкие символы кодируются с помощью 4 байтов.

Например, свойство length считает, что здесь два символа:

Свойство length воспринимает 4-байтовый символ как два символа по 2 байта. Это неверно, потому что эти два символа должны восприниматься как единое целое (так называемая «суррогатная пара»).

Регулярки также по умолчанию воспринимают 4-байтные «длинные символы» как пары 2-байтных. В отличие от строк, у регулярных выражений есть специальный флаг u, который исправляет эту проблему.

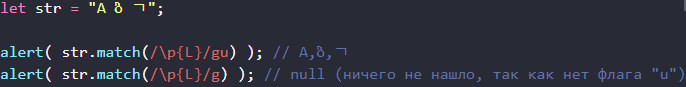
### Юникодные свойства \p{…}

Каждому символу в Юникоде соответствует множество свойств. Они описывают к какой «категории» относится символ, содержат различную информацию о нём.

Например, свойство Letter у символа означает, что это буква какого-то алфавита. А Number означает, что это цифра.

В регулярках можно искать символ с заданным свойством, указав его в \p{…}. Для таких регулярных выражений обязательно использовать флаг u.

В примере ниже будут найдены английская, грузинская и корейская буквы:

основные категории символов и их подкатегории с примерами:

https://learn.javascript.ru/regexp-unicode

## Якоря: начало строки ^ и конец $

Каретка ^ означает совпадение с началом текста, а доллар $ – с концом.

К примеру, давайте проверим начинается ли текст с Mary:

Шаблон ^Mary означает: «начало строки, затем Mary»

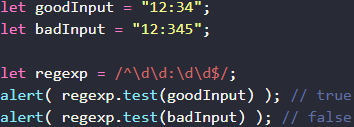
Аналогично можно проверить, кончается ли строка словом snow при помощи snow$:



### Проверка на полное совпадение

Оба якоря вместе ^...$ часто используются для проверки, совпадает ли строка с шаблоном полностью.

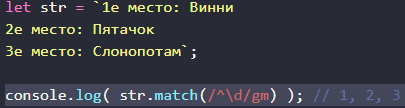
Проверим, что строка является временем в формате 12:34:

Якоря ^ и $ – это проверки. У них нулевая ширина.

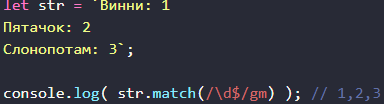
## Многострочный режим якорей ^ $, флаг "m"

Многострочный режим включается флагом m. Он влияет только на поведение ^ и $. В многострочном режиме они означают не только начало/конец текста, но и начало/конец каждой строки в тексте.

В примере ниже текст состоит из нескольких строк. Шаблон /^\d/gm берёт цифру с начала каждой строки:

 без флага m было бы найдено только первое число.

Символ доллара $ ведёт себя аналогично. Регулярное выражение \d$ ищет последнюю цифру в каждой строке:



## Граница слова: \b

Граница слова \b – проверка, как ^ и $. Проверяет, что позиция в строке является границей слова.

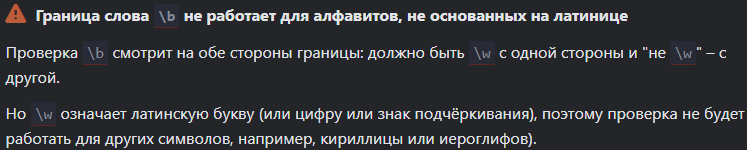
Есть три вида позиций, которые являются границами слова:

* Начало текста, если его первый символ \w
* Позиция внутри текста, если слева находится \w, а справа – не \w, или наоборот
* Конец текста, если его последний символ \w

Например, \bJava\b будет найдено в строке Hello, Java!, где Java – отдельное слово, но не будет найдено в строке Hello, JavaScript!

Можно использовать \b не только со словами, но и с цифрами:

\b\d\d\b ищет отдельно стоящие двузначные числа. Оно требует, чтобы и до и после \d\d были символы, отличные от \w



## Экранирование, специальные символы

Обратная косая черта \ используется для обозначения классов символов, например \d. Есть и другие специальные символы, например [ ] \ ^ $ . | ? \* + ( ).

Допустим, мы хотим найти буквально точку. Не «любой символ», а именно точку. Чтобы использовать специальный символ как обычный, добавьте к нему обратную косую черту: \. Пример:



В приведённом ниже примере ищется строка "g()":



### Косая черта

Символ косой черты '/', в JavaScript используется для открытия и закрытия регулярки поэтому мы должны экранировать его.



Если через RegExp тогда нам не нужно экранировать косую черту.

### new RegExp

Если мы создаём регулярное выражение с помощью new RegExp, то нам не нужно учитывать /, но нужно другое экранирование.

Например, такой поиск не работает:



Причина в том, что символы обратной косой черты «съедаются» строкой. Как вы помните, обычные строки имеют свои специальные символы, такие как \n, и для экранирования используется обратная косая черта.

Вот как воспринимается строка «\d.\d»:

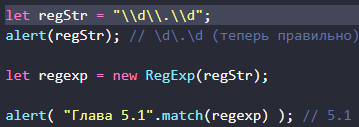


Строковые кавычки «съедают» символы обратной косой черты для себя, например:

* \n – становится символом перевода строки,
* \u1234 – становится символом Юникода с указанным номером,
* …А когда нет особого значения: как например для \d или\z, обратная косая черта просто удаляется.

Таким образом, new RegExp получает строку без обратной косой черты. Вот почему поиск не работает!

Чтобы исправить это, нам нужно удвоить обратную косую черту, потому что строковые кавычки превращают \\ в \:



## Наборы и диапазоны [...]