

Регулярные выражения

The logo for Regular Expressions, featuring the word "Reg" in purple and "Ex" in orange, both in a bold, sans-serif font.

Regular Expression

```
/h[a4@](((c<)((k)|(\<)))|((k)|(\<)))(x))\s+\  
((d)|([t\+])h)[3ea4@]\s+p[l1][a4@]n[3e][t\+]/i
```

©2006 FTS Conventures - www.ftskonventures.com

Регулярное выражение — шаблон (образец), по которому выполняется поиск соответствующего ему фрагмента текста

Регулярные выражения предназначены для обработки текстовой информации и обеспечивают:

- эффективный **поиск** в тексте по заданному шаблону;
- **редактирование**, **замену** и **удаление** подстрок;
- формирование **ИТОВОВЫХ ОТЧЕТОВ** по результатам работы с текстом.

История

Регулярные выражения изначально появилась в среде UNIX и использовались в языке программирования Perl.

Разработчики из Microsoft перенесли регулярные выражения в Windows, где теперь они поддерживаются множеством классов **.NET** из пространства имен **System.Text.RegularExpressions.**

Элементы языка регулярных выражений — краткий справочник
[https://msdn.microsoft.com/ru-ru/library/az24scfc\(v=vs.110\).aspx](https://msdn.microsoft.com/ru-ru/library/az24scfc(v=vs.110).aspx)

Язык описания регулярных выражений

Язык описания регулярных выражений состоит из символов двух видов: **обычных** и **метасимволов**.

- **Обычный символ** представляет в выражении сам себя.
- **Метасимвол** — **специальные символы, задающие команды** :
 - ❖ **класс символов** (например, любая цифра **\d** или буква **\w**)
 - ❖ **уточняющий символ** (например, **^**).
 - ❖ **повторитель** (например, **+**).

Примеры:

- выражение для поиска в тексте фрагмента «Вася» записывается с помощью четырех обычных символов «Вася»
- выражение для поиска двух цифр, идущих подряд, состоит из двух метасимволов «\d\d»
- выражение для поиска фрагментов вида «Вариант 1», «Вариант 2», ..., «Вариант 9» имеет вид «Вариант \d»
- выражение для поиска фрагментов вида «Вариант 1», «Вариант 23», «Вариант 719», ..., имеет вид «Вариант \d+»

Метасимволы - классы символов

	Описание	Пример
.	любой символ, кроме \n	c.T соответствует фрагментам cat, cut, clt, c{t и т.д.
[]	любой одиночный символ из последовательности внутри скобок.	c[au1]t соответствует фрагментам cat, cut и clt c[a-z]t соответствует фрагментам cat, cbt, cct, cdt,
[^]	любой одиночный символ, не входящий в последовательность внутри скобок.	c[^au1]t соответствует фрагментам cbt, c2t, cXt c[^a-zA-Z]t соответствует фрагментам cit, clt, c4t, c3t

\w	любой алфавитно-цифровой символ, то есть символ из множества прописных и строчных букв и десятичных цифр	c\wt соответствует фрагментам cat, cut, c1t, cЮt Не соответствует c{t, c;t и т.д.
\W	любой не алфавитно-цифровой символ, то есть символ, не входящий в множество прописных и строчных букв и десятичных цифр	c\Wt соответствует фрагментам c{t, c;t, c t и т.д. Не соответствует cat, cut, c1t, cЮt
\s	любой пробельный символ, например, пробел, табуляция (\t, \v), перевод строки (\n, \r), новая страница (\f)	\s\w\w\w\s соответствует любому слову из трех букв (цифр), окруженному пробельными символами
\S	любой не пробельный символ, то есть символ, не входящий в множество пробельных	\s\S\S\s соответствует любым двум непробельным символам, окруженным пробельными.
\d	любая десятичная цифра	c\dt соответствует фрагментам c1t, c2t, ..., c9t
\D	любой символ, не являющийся десятичной цифрой	c\Dt не соответствует фрагментам c1t, c2t, ..., c9t. ⁷

Символы повторения

{n,m}	Соответствует предшествующему шаблону, повторенному не менее n и не более m раз	ca{2,4}t соответствует фрагментам caat, caaat и caaaat
{n,}	Соответствует предшествующему шаблону, повторенному n или более раз	ca{3,}t соответствует фрагментам caaat, caaaat, caaaaaaaaaaaat и т.д.
{n}	Соответствует в точности n экземплярам предшествующего шаблона	ca{3}t соответствует фрагменту caaat (cat){2} соответствует фрагменту catcat
?	Соответствует нулю или одному экземпляру предшествующего шаблона; предшествующий шаблон является необязательным	Эквивалентно {0,1}
+	Соответствует одному или более экземплярам предшествующего шаблона	Эквивалентно {1,}
*	Соответствует нулю или более экземплярам предшествующего шаблона	Эквивалентно {0,}

Якорные символы регулярных выражений

^	Соответствует началу строкового выражения или началу строки при многострочном поиске.	^Hello	"Hello, world", но не "Ok, Hello world" т.к. в этой строке слово "Hello" находится не в начале
\$	Соответствует концу строкового выражения или концу строки при многострочном поиске.	Hello\$	"World, Hello"
\b	Соответствует границе слова, т.е. соответствует позиции между символом \w и символом \W или между символом \w и началом или концом строки.	\b(my)\b	В строке "Hello my world" выберет слово "my"
\B	Соответствует позиции, не являющейся границей слов.	\B(ld)\b	Соответствие найдется в слове "World", но не в слове "ld"

RFC2822 Mail Address: regexp-based address validation

[illegible]

кот		КОТ - окто КОТ киткоооткат толчет в КОТ ле мак
ко+т	«о» от 1 до n	КОТ -окто КОТ кит КОО откат толчет в КОТ ле мак
ко*т	«о» от 0 до n	КОТ -окто КОТ кит КОО откат толчет в КОТ ле мак
ко?т	«о» от 0 или 1	КОТ -окто КОТ киткоооткат толчет в КОТ ле мак
ко{3}т	«о» 3 раза	КОТ КООТ КОООТ коооот коооооот
ко{3,}т	«о» не меньше 3-х раз	КОТ КООТ КОООТ КООООТ КООООООТ
ко{,3}т	«о» не больше 3-х раз	КОТ КООТ КОООТ коооот коооооот
ко{3,5}т	«о» от 3-х до 5-ти	КОТ КООТ КОООТ КООООТ коооооот
к[оаыи]т	между «к» и «т» - «о,а,ы,и»	КОТ -окто КОТ киткооот КАТ толчет в КОТ ле мак
к(о а ы и)т	между «к» и «т» - «о,а,ы,и»	КОТ -окто КОТ киткооот КАТ толчет в КОТ ле мак
к(о ро апо и)т	между «к» и «т» - «о,ро,апо,и»	КОТ , КРОТ , КАПОТ и КИТ
к[оаыи]+т	между «к» и «т» - «о,а,ы,и» от 1 до n	КОТ -окто КОТ кит КОО от КАТ толчет в КОТ ле мак
к[а-я]+т	между «к» и «т» - гласные буквы от 1 до n	КОТ -окто КОТ кит КОО от КАТ толчет в КОТ ле мак
\bkот\b	\b – граница слова	КОТ октокот киткоооткат толчет в котле мак
\w*кот\b	«кот» - справа	КОТ -окто КОТ киткоооткат толчет в котле мак
\bkот\w*	«кот» - слева	КОТ -октокот киткоооткат толчет в КОТ ле мак

[illegible]

Примеры простых регулярных выражений

- целое число (возможно, со знаком):

`[-+]? \d+`

- вещественное число (может иметь знак и дробную часть, отделенную точкой):

`[-+]? \d+ \. ? \d*`

- российский номер автомобиля (упрощенно):

`[A-Z] \d{3} [A-Z]{2} \d \d RU S`

- ip-адрес (упрощенно):

`(\d{1,3} \.){3} \d{1,3}`

Дата в формате YYYY-MM-DD:

[0-9]{4}-(0[1-9]|1[012])-(0[1-9]|1[0-9]|2[0-9]|3[01])

Номер кредитки:

[0-9]{13,16}

Имя пользователя (с ограничением 2-20 символов, которыми могут быть буквы и цифры, первый символ обязательно буква):

^[a-zA-Z][a-zA-Z0-9_\.]{1,20}\$

Домен (например abcd.com):

^([a-zA-Z0-9]([a-zA-Z0-9\-.]{0,61}[a-zA-Z0-9])?\.)+[a-zA-Z]{2,6}\$

IPv4

((25[0-5]|2[0-4]\d|[01]?\d\d?)\.){3}(25[0-5]|2[0-4]\d|[01]?\d\d?)

Пароль (Строчные и прописные латинские буквы, цифры, спецсимволы. Минимум 8 символов):

(?=^.{8,}\$)((?=.*\d)(?=.*\W+))(?![\.\n])(?=.*[A-Z])(?=.*[a-z]).*\$

Целые числа и числа с плавающей точкой (разделитель точка):

\-?\d+(\.\d{0,})?

Поддержка регулярных выражений в .NET

Для поддержки регулярных выражений в библиотеку **.NET** **включены классы**, объединенные в пространство имен

System.Text.RegularExpressions

- Основной класс – **Regex**. Он реализует подсистему обработки регулярных выражений.
- Подсистеме требуется предоставить:
 - **Шаблон** (регулярное выражение), соответствия которому требуется найти в тексте.
 - **Текст**, который требуется проанализировать с помощью шаблона.

Использование класса Regex

Обработчик регулярных выражений выполняет синтаксический **разбор** и **компиляцию** регулярного выражения, а также операции, **сопоставляющие** шаблон регулярного выражения с входной строкой.

```
string pattern = @"^\d{3}-\d{2}-\d{4}";  
string[] values = { "123-22-3789", "123-2-3333" };
```

Обработчик можно использовать одним из двух способов:

- С помощью вызова статических методов класса [Regex](#). Параметры метода содержат входную строку и шаблон регулярного выражения.

```
if (Regex.IsMatch(value, pattern)){ ...}
```

- С помощью создания объекта [Regex](#) посредством передачи регулярного выражения в конструктор класса.

```
Regex regEx = new Regex(pattern, RegexOptions.IgnoreCase);  
if (regEx.IsMatch(value)) { ...}
```


Член	Описание
CultureInvariant	Указывает игнорирование региональных языковых различий
ExplicitCapture	Модифицирует способ поиска соответствия, обеспечивая только буквальное соответствие
IgnoreCase	Находит совпадения независимо от регистра, т.е. прописными или строчными буквами в строке написано слово.
IgnorePatternWhitespace	Удаляет из шаблона неизбежные пробелы и включает комментарии помеченные «#»
Multiline	Изменяет значение символов ^ и \$ так, что они применяются к началу и концу каждой строки, а не только к началу и концу всего входного текста
RightToLeft	Предписывает читать входную строку справа налево вместо направления по умолчанию — слева направо (что удобно для некоторых азиатских и других языков, которые читаются в таком направлении)
Singleline	Специфицирует однострочный режим, в котором точка (.) символизирует соответствие любому символу

Методы класса Regex

позволяют выполнять следующие действия:

- Определить, **встречается ли** во входном тексте шаблон регулярного выражения (метод **IsMatch()**).
- **Извлечь** из текста одно или все вхождения, соответствующие шаблону регулярного выражения (методы **Match()** или **Matches()** (возврат MatchCollection)).
- **Заменить** текст, соответствующий шаблону регулярного выражения (метод **Replace()**).
- **Разделить** строку на массив строк (метод **Split()**).

Примеры использования класса Regex

```
// поиск номера соц страхования (его форма XXX-XX-XXXX)
string[] values = { "123-22-3789", "123-2-3333" };
string pattern = @"^\d{3}-\d{2}-\d{4}";
foreach (string value in values)
{
    if (Regex.IsMatch(value, pattern))
        Console.WriteLine("{0} номер соц страхования!", value);
    else
        Console.WriteLine("{0}: номер не опознан!", value);
}

// ВТОРОЙ СПОСОБ...
Regex regEx = new Regex(pattern, RegexOptions.IgnoreCase);
foreach (string value in values)
{
    if (regEx.IsMatch(value))
        Console.WriteLine("{0} номер соц страхования!", value);
    else
        Console.WriteLine("{0}: номер не опознан!", value);
}
```

```
123-22-3789 номер соц страхования!
123-2-3333: номер не опознан!
```

Примеры использования класса Regex

// поиск нескольких совпадений в строке

```
Console.WriteLine("Пример 1");
string pattern = "abc";
string input = "abc123abc456abc789";
foreach (Match match in Regex.Matches(input, pattern))
{ Console.WriteLine("{0} найдено! Индекс - {1}.",
                    match.Value, match.Index);
}
```

```
Console.WriteLine("Использование Match");
Match match2 = Regex.Match(input, pattern);
while (match2.Success)
{
    Console.WriteLine("{0} найдено! Индекс - {1}.",
                      match2.Value, match2.Index);
    match2 = match2.NextMatch();
}
```

```
Пример 1
abc найдено! Индекс - 0.
abc найдено! Индекс - 6.
abc найдено! Индекс - 12.
Использование Match
abc найдено! Индекс - 0.
abc найдено! Индекс - 6.
abc найдено! Индекс - 12.
```

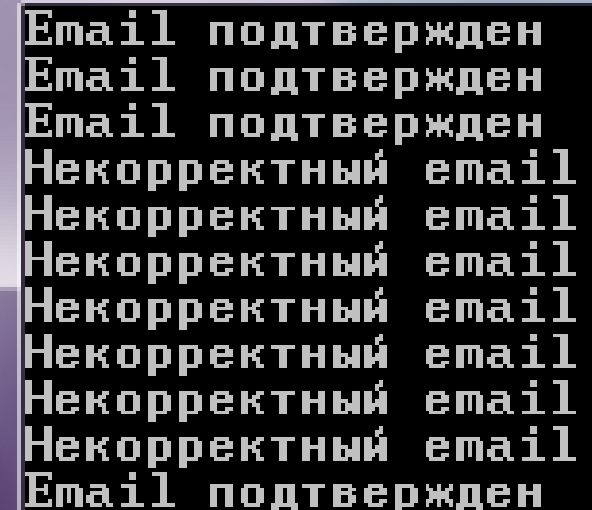
Примеры использования класса Regex

```
Console.WriteLine("Пример 2");  
string pattern2 = @"\b91*9*\b";  
string input2 = "99 95 919 929 9119 9219 999 9919 91119";  
foreach (Match match in Regex.Matches(input2, pattern2))  
    Console.WriteLine("{0} найдено! Индекс - {1}.",  
                      match.Value, match.Index);
```

```
Пример 2  
99 найдено! Индекс - 0.  
919 найдено! Индекс - 6.  
9119 найдено! Индекс - 14.  
999 найдено! Индекс - 24.  
91119 найдено! Индекс - 33.
```

Примеры использования метода IsMatch

```
string pattern = @"^([a-z0-9_-]+\.)*[a-z0-9_-]+@[a-z0-9_-]+(\.[a-z0-9_-]+)*\.[a-z]{2,6}$";
string[] emailAddresses = {
    "david@pros.com", "d.j@server1.proseware.com",
    "jones@ms1.proseware.com",
    "j.@server1.proseware.com", "j@proseware.com9",
    "js#internal@proseware.com",
    "j..s@proseware.com", "js*@proseware.com", "js@proseware..com",
    "js@proseware.com9", "j.s@server1.proseware.com" };
foreach (string emailAddress in emailAddresses)
{
    if (Regex.IsMatch(emailAddress, pattern))
        Console.WriteLine("Email подтвержден");
    else
        Console.WriteLine("Некорректный email");
}
Console.ReadKey();
```



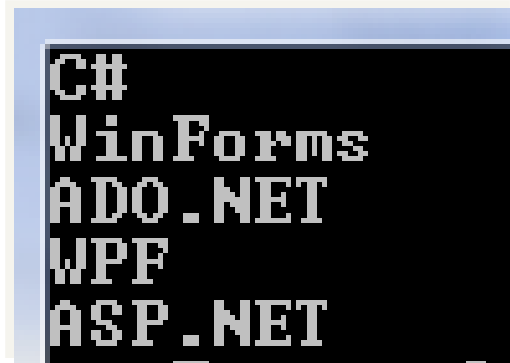
```
Email подтвержден
Email подтвержден
Email подтвержден
Некорректный email
Некорректный email
Некорректный email
Некорректный email
Некорректный email
Некорректный email
Некорректный email
Некорректный email
Email подтвержден
```

<http://tealeaf.mmailm.lclients.ru/page/2/12>

<http://tealeaf.mmailm.lclients.ru/page/2/11>

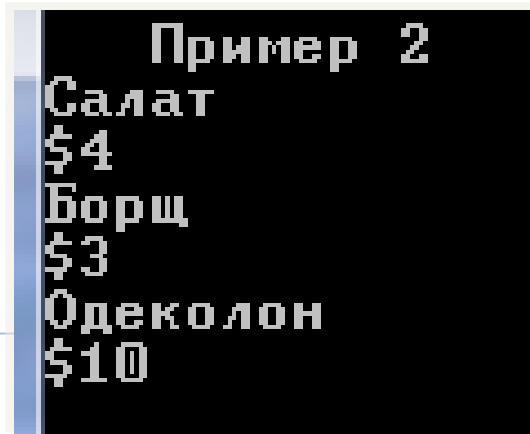
Примеры использования метода Split

```
// Помещает элементы нумерованного списка в массив строк
string input = "1. C# 2. WinForms 3.ADO.NET 4. WPF 5. ASP.NET";
string pattern = @"\d{1,2}\.\s?"; // ? - для 3.ADO.NET
foreach (string item in Regex.Split(input, pattern))
{
    if (!String.IsNullOrEmpty(item))
        Console.WriteLine(item);
}
```



```
C#
WinForms
ADO.NET
WPF
ASP.NET
```

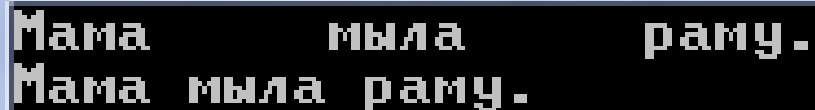
```
Console.WriteLine("    Пример 2");
// Разбиение текста на слова!
string text = "Салат - $4, Борщ -$3, Одеколон - $10.";
string pattern2 = "[-,.]+";
Regex r = new Regex(pattern2);
string[] words = r.Split(text);
foreach (string word in words)
{
    Console.WriteLine(word);
}
```



```
Пример 2
Салат
$4
Борщ
$3
Одеколон
$10
```

Примеры использования метода Replace

```
string s = "Мама      мыла      раму. ";  
string pattern = @"\\s+";  
string target = " ";  
Regex regex = new Regex(pattern);  
string result = regex.Replace(s, target);  
Console.WriteLine(s);  
Console.WriteLine(result);
```



```
Мама      мыла      раму.  
Мама мыла раму.
```


Группирование

Группирование (с помощью круглых скобок) применяется во многих случаях:

- требуется задать повторитель не для отдельного символа, а для последовательности;
- для запоминания фрагмента, совпавшего с выражением, заключенным в скобки, в некоторой переменной. Имя переменной задается в угловых скобках или апострофах:

(?**<имя_переменной>**фрагмент_выражения)

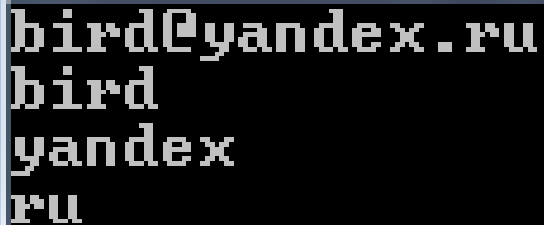
например: номера телефонов в виде **nnn-nn-nn** запоминаются в переменной num:

(?**<num>** \d\d\d-\d\d-\d\d)

Примеры использования группирования

```
// Создадим регулярное выражения для поиска
// простых адресов e-mail: (\w+)@(\w+).(\w+)
Regex regEx = new Regex(@"(\w+)@(\w+).(\w+)");
Match m = regEx.Match("Мой почтовый адрес bird@yandex.ru");

Console.WriteLine(m.Groups[0]); // выведет bird@yandex.ru"
Console.WriteLine(m.Groups[1]); // выведет bird
Console.WriteLine(m.Groups[2]); // выведет yandex
Console.WriteLine(m.Groups[3]); // выведет ru
Console.ReadKey();
```

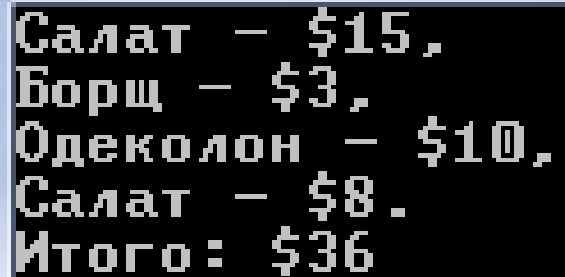


```
bird@yandex.ru
bird
yandex
ru
```

Примеры использования группирования

```
string text = "Салат - $15,  
               Борщ - $3,  
               Одеколон - $10,  
               Салат - $8.";

string pattern = @"(\w+) - \$(\d+)[.,]";
Regex regEx = new Regex(pattern);
Match match = regEx.Match(text);
int total = 0;
while (match.Success)
{
    Console.WriteLine(match);
    total += int.Parse(match.Groups[2].ToString());
    match = match.NextMatch();
}
Console.WriteLine("Итого: $" + total);
```



```
Салат - $15,  
Борщ - $3,  
Одеколон - $10,  
Салат - $8.  
Итого: $36
```

Просмотр вперед и назад

Регулярное выражение, которое учитывает подстроки находящиеся до либо после результата, но в сам результат не попадающие.

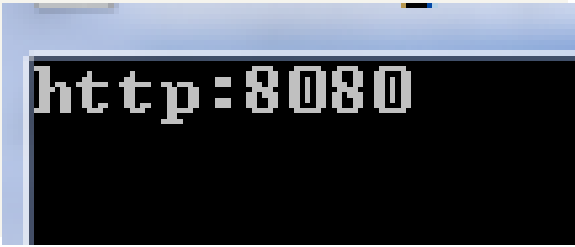
- (?=шаблон) - позитивный просмотр вперед
- (?!шаблон) - негативный просмотр вперед
- (?<=шаблон) - позитивный просмотр назад
- (?<!шаблон) - негативный просмотр назад

Примеры просмотров вперед и назад

два(?=\s+три)	Слово два справа от которого находится слово три	один два <u>три</u> два один два один.	два
два(?!\s+три)	Слово два справа от которого нет слова три	один два три два <u>один</u> два <u>один</u> .	два два
(?<=три\s+)два	Слово два слева от которого находится слово три	один два <u>три</u> два один два один.	два
(?<!три\s+)два	Слово два слева от которого нет слова три	<u>один</u> два три два <u>один</u> два один.	два два

Примеры использования класса Regex

```
string url = "http://www.contoso.com:8080/letters/readme.html";  
Regex r = new Regex(@"^(?<proto>\w+):\/\/[^\s/]+?(?<port>:\d+)?/");  
Match m = r.Match(url);  
if (m.Success)  
    Console.WriteLine(r.Match(url).Result("${proto}${port}"));
```



http:8080

^(?<proto>\w+)://[^\s/]+?(?<port>:\d+)?/

^

Соответствие должно обнаруживаться в начале строки.

(?<proto>\w+)

Совпадение с одним или несколькими символами слова. Эта группа должна получить имя proto.

://

Соответствует двоеточию, за которым следуют две косые черты.

[^\s/]+?

Соответствует одному или нескольким вхождениям (но как можно меньшему числу) любого символа, отличного от косой черты.

(?<port>:\d+)?

Соответствует вхождениям в количестве 0 или 1 двоеточия, за которым следует одна или несколько цифр. Эта группа должна получить имя port.

/

Соответствует косой черте.

Сайты для проверки регулярных выражений

- <http://www.regexr.com/>
- <https://regex101.com>