



Tproger

[Алгоритмы](#)[Для новичков](#)[Gamedev](#)[Mobile](#)[Web](#)[Linux](#)[Android](#)[C++](#)[Язык Си](#)[C#](#)[Java](#)

Использование регулярных выражений в Python для новичков

10 июня 2015 в 18:27, Переводы  10 минут  35 440

 This is interesting

 1K



В последние годы языки общего назначения стали чаще использоваться для анализа данных. Разработчики и организации используют Python или Javascript для решения своих задач. И в этом им помогают регулярные выражения. Они — незаменимый инструмент для упорядочивания, причесывания, поиска или извлечения текстовых данных.

Все это делает регулярные выражения полезными для изучения.

В этой статье мы рассмотрим примеры использования и применения регулярных выражений. Они часто используются программистами в различных языках — Perl, C++, Java. Мы будем использовать Python. Ближе к концу мы также посмотрим на некоторые реальные задачи, решаемые с их помощью.

Что такое регулярные выражения и как их использовать?

Говоря простым языком, регулярное выражение — это последовательность символов, используемая для поиска и замены текста в строке или файле. Как уже было упомянуто, их поддерживает множество языков общего назначения: [Python](#), [Perl](#), [R](#). Так что изучение регулярных выражений рано или поздно пригодится.

Регулярные выражения используют два типа символов:

- специальные символы: как следует из названия, у этих символов есть специальные значения, например, `*` означает «любой символ»;
- литералы (например: `a`, `b`, `1`, `2` и т. д.).

В Python для работы с регулярными выражениями есть модуль `re`. Для использования его нужно импортировать:

```
1 import re
```

Чаще всего регулярные выражения используются для:

- поиска в строке;
- разбиения строки на подстроки;
- замены части строки.

Давайте посмотрим на методы, которые предоставляет библиотека `re` для этих задач. Вот наиболее часто используемые из них:

- `re.match()`
- `re.search()`
- `re.findall()`
- `re.split()`
- `re.sub()`
- `re.compile()`

Рассмотрим их подробнее.

`re.match(pattern, string):`

Этот метод ищет по заданному шаблону в начале строки. Например, если мы вызовем метод `match()` на строке «AV Analytics AV» с шаблоном «AV», то он завершится успешно. Однако если мы будем искать «Analytics», то результат будет отрицательный. Давайте посмотрим на его работу:

```
1 import re
2 result = re.match(r'AV', 'AV Analytics Vidhya AV')
3 print result
4
5 Результат:
```

Искомая подстрока найдена. Чтобы вывести ее содержимое, используем метод `group()`. (Мы используем «r» перед строкой шаблона, чтобы показать, что это «сырая» строка в Python).

```
1 result = re.match(r'AV', 'AV Analytics Vidhya AV')
2 print result.group(0)
```

```
3
4 Результат:
5 AV
```

Теперь попробуем найти «Analytics» в данной строке. Поскольку строка начинается на «AV», метод вернет `None`:

```
1 result = re.match(r'Analytics', 'AV Analytics Vidhya AV')
2 print result
3
4 Результат:
5 None
```

Также есть методы `start()` и `end()` для того, чтобы узнать начальную и конечную позицию найденной строки.

```
1 result = re.match(r'AV', 'AV Analytics Vidhya AV')
2 print result.start()
3 print result.end()
4
5 Результат:
6 0
7 2
```

Эти методы иногда очень полезны для работы со строками.

`re.search(pattern, string):`

Этот метод похож на `match()`, но он ищет не только в начале строки. В отличие от предыдущего, `search()` вернет объект, если мы попытаемся найти «Analytics».

```
1 result = re.search(r'Analytics', 'AV Analytics Vidhya AV')
2 print result.group(0)
```

```
3
4 Результат:
5 Analytics
```

Метод `search()` ищет по всей строке, но возвращает только первое найденное совпадение.

`re.findall(pattern, string):`

Этот метод возвращает список всех найденных совпадений. У метода `findall()` нет ограничений на поиск в начале или конце строки. Если мы будем искать «AV» в нашей строке, он вернет все вхождения «AV». Для поиска рекомендуется использовать именно `findall()`, так как он может работать и как `re.search()`, и как `re.match()`.

```
1 result = re.findall(r'AV', 'AV Analytics Vidhya AV')
2 print result
3
4 Результат:
5 ['AV', 'AV']
```

`re.split(pattern, string, [maxsplit=0]):`

Этот метод разделяет строку по заданному шаблону.

```
1 result = re.split(r'y', 'Analytics')
2 result
3
4 Результат:
5 ['Anal', 'tics']
```

В примере мы разделили слово «Analytics» по букве «y». Метод `split()` принимает также аргумент `maxsplit` со значением по умолчанию, равным 0. В данном случае он разделит строку столько раз, сколько возможно, но если указать этот аргумент, то разделение будет

произведено не более указанного количества раз. Давайте посмотрим на примеры:

```
1 result = re.split(r'i', 'Analytics Vidhya')
2 print result
3
4 Результат:
5 ['Analyt', 'cs V', 'dhya'] # все возможные участки.
```

```
1 result = re.split(r'i', 'Analytics Vidhya',maxsplit=1)
2 result
3
4 Результат:
5 ['Analyt', 'cs Vidhya']
```

Мы установили параметр `maxsplit` равным 1, и в результате строка была разделена на две части вместо трех.

`re.sub(pattern, repl, string):`

Этот метод ищет шаблон в строке и заменяет его на указанную подстроку. Если шаблон не найден, строка остается неизменной.

```
1 result = re.sub(r'India', 'the World', 'AV is largest Analytics community of India')
2 result
3
4 Результат:
5 'AV is largest Analytics community of the World'
```

`re.compile(pattern, repl, string):`

Мы можем собрать регулярное выражение в отдельный объект, который может быть использован для поиска. Это также избавляет от переписывания одного и того же

выражения.

```
1 import re
2 pattern = re.compile('AV')
3 result = pattern.findall('AV Analytics Vidhya AV')
4 print result
5 result2 = pattern.findall('AV is largest analytics community of India')
6 print result2
7
8 Результат:
9 ['AV', 'AV']
10 ['AV']
```

До сих пор мы рассматривали поиск определенной последовательности символов. Но что, если у нас нет определенного шаблона, и нам надо вернуть набор символов из строки, отвечающий определенным правилам? Такая задача часто стоит при извлечении информации из строк. Это можно сделать, написав выражение с использованием специальных символов. Вот наиболее часто используемые из них:

Оператор	Описание
.	Один любой символ, кроме новой строки \n.
?	0 или 1 вхождение шаблона слева
+	1 и более вхождений шаблона слева
*	0 и более вхождений шаблона слева
\w	Любая цифра или буква (\W — все, кроме буквы или цифры)
\d	Любая цифра [0-9] (\D — все, кроме цифры)
\s	Любой пробельный символ (\S — любой непробельный символ)

\b	Граница слова
[...]	Один из символов в скобках ([^...] — любой символ, кроме тех, что в скобках)
\	Экранирование специальных символов (\. означает точку или \+ — знак «плюс»)
^ и \$	Начало и конец строки соответственно
{n,m}	От n до m вхождений ({,m} — от 0 до m)
a b	Соответствует a или b
()	Группирует выражение и возвращает найденный текст
\t, \n, \r	Символ табуляции, новой строки и возврата каретки соответственно

Больше информации по специальным символам, таким как (), | и др., можно найти на странице документации: <https://docs.python.org/2/library/re.html>).

Теперь давайте посмотрим на примеры использования регулярных выражений.

Примеры использования регулярных выражений

Задача 1: Вернуть первое слово из строки

Сначала попробуем вытащить каждый символ (используя `\w`)

```

1 import re
2 result = re.findall(r'.', 'AV is largest Analytics community of India')
3 print result
4
5 Результат:
6 ['A', 'V', ' ', 'i', 's', ' ', 'l', 'a', 'r', 'g', 'e', 's', 't', ' ', 'A', 'n', 'a', 'l', 'y', 't', 'i', 'c', 's', ' ', 'c', 'o', 'm', 'm', 'u', 'n', 'i', 't', 'y', ' ', 'o', 'f', ' ', 'I', 'n', 'd', 'i', 'a']

```

Для того, чтобы в конечный результат не попал пробел, используем вместо `.` `\w`.

```
1 result = re.findall(r'\w', 'AV is largest Analytics community of India')
2 print result
3
4 Результат:
5 ['A', 'V', 'i', 's', 'l', 'a', 'r', 'g', 'e', 's', 't', 'A', 'n', 'a', 'l', 'y', 't', 'i', 'c']
```

Теперь попробуем достать каждое слово (используя `*` или `+`)

```
1 result = re.findall(r'\w*', 'AV is largest Analytics community of India')
2 print result
3
4 Результат:
5 ['AV', '', 'is', '', 'largest', '', 'Analytics', '', 'community', '', 'of', '', 'India', '']
```

И снова в результат попали пробелы, так как `*` означает «ноль или более символов». Для того, чтобы их убрать, используем `+`:

```
1 result = re.findall(r'\w+', 'AV is largest Analytics community of India')
2 print result
3
4 Результат:
5 ['AV', 'is', 'largest', 'Analytics', 'community', 'of', 'India']
```

Теперь вытащим первое слово, используя `^`:

```
1 result = re.findall(r'^\w+', 'AV is largest Analytics community of India')
2 print result
3
4 Результат:
5 ['AV']
```

Если мы используем `$` вместо `^`, то мы получим последнее слово, а не первое:

```
1 result = re.findall(r'\w+$', 'AV is largest Analytics community of India')
```

```
2 print result
3
4 Результат:
5 ['India']
```

Задача 2: Вернуть первые два символа каждого слова

Вариант 1: используя `\w`, вытащить два последовательных символа, кроме пробельных, из каждого слова:

```
1 result = re.findall(r'\w\w', 'AV is largest Analytics community of India')
2 print result
3
4 Результат:
5 ['AV', 'is', 'la', 'rg', 'es', 'An', 'al', 'yt', 'ic', 'co', 'mm', 'un', 'it', 'of', 'In', 'd']
```

Вариант 2: вытащить два последовательных символа, используя символ границы слова (`\b`):

```
1 result = re.findall(r'\b\w\w', 'AV is largest Analytics community of India')
2 print result
3
4 Результат:
5 ['AV', 'is', 'la', 'An', 'co', 'of', 'In']
```

Задача 3: вернуть список доменов из списка адресов электронной почты

Давайте снова рассмотрим решение пошагово. Сначала вернем все символы после «@»:

```
1 result = re.findall(r'@\w+', 'abc.test@gmail.com, xyz@test.in, test.first@analyticsvidhya.com')
2 print result
3
```

```
4 Результат:
5 ['@gmail', '@test', '@analyticsvidhya', '@rest']
```

Как видим, части «.com», «.in» и т. д. не попали в результат. Изменим наш код:

```
1 result = re.findall(r'@\w+.\w+', 'abc.test@gmail.com, xyz@test.in, test.first@analyticsvidhya')
2 print result
3
4 Результат:
5 ['@gmail.com', '@test.in', '@analyticsvidhya.com', '@rest.biz']
```

Второй вариант — вытащить только домен, используя группировку — `()`:

```
1 result = re.findall(r'@\w+.( \w+)', 'abc.test@gmail.com, xyz@test.in, test.first@analyticsvidhya')
2 print result
3
4 Результат:
5 ['com', 'in', 'com', 'biz']
```

Задача 4: Извлечь дату из строки

Используем `\d` для извлечения цифр.

```
1 result = re.findall(r'\d{2}-\d{2}-\d{4}', 'Amit 34-3456 12-05-2007, XYZ 56-4532 11-11-2011, Amit 12-01-2009')
2 print result
3
4 Результат:
5 ['12-05-2007', '11-11-2011', '12-01-2009']
```

Для извлечения только года нам опять помогут скобки:

```
1 result = re.findall(r'\d{2}-\d{2}-(\d{4})', 'Amit 34-3456 12-05-2007, XYZ 56-4532 11-11-2011, Amit 12-01-2009')
2 print result
3
4 Результат:
5 ['2007', '2011', '2009']
```

Задача 5: Извлечь все слова, начинающиеся на гласную

Для начала вернем все слова:

```
1 result = re.findall(r'\w+', 'AV is largest Analytics community of India')
2 print result
3
4 Результат:
5 ['AV', 'is', 'largest', 'Analytics', 'community', 'of', 'India']
```

А теперь — только те, которые начинаются на определенные буквы (используя `[]`):

```
1 result = re.findall(r'[aeiouAEIOU]\w+', 'AV is largest Analytics community of India')
2 print result
3
4 Результат:
5 ['AV', 'is', 'argest', 'Analytics', 'ommunity', 'of', 'India']
```

Выше мы видим обрезанные слова «argest» и «ommunity». Для того, чтобы убрать их, используем `\b` для обозначения границы слова:

```
1 result = re.findall(r'\b[aeiouAEIOU]\w+', 'AV is largest Analytics community of India')
2 print result
3
4 Результат:
5 ['AV', 'is', 'Analytics', 'of', 'India']
```

Также мы можем использовать `^` внутри квадратных скобок для инвертирования группы:

```
1 result = re.findall(r'\b[^aeiouAEIOU]\w+', 'AV is largest Analytics community of India')
2 print result
3
4 Результат:
5 [' is', ' largest', ' Analytics', ' community', ' of', ' India']
```

В результат попали слова, «начинающиеся» с пробела. Уберем их, включив пробел в диапазон в квадратных скобках:

```
1 result = re.findall(r'\b[^aeiouAEIOU ]\w+', 'AV is largest Analytics community of India')
2 print result
3
4 Результат:
5 ['largest', 'community']
```

Задача 6: Проверить телефонный номер (номер должен быть длиной 10 знаков и начинаться с 8 или 9)

У нас есть список телефонных номеров, и нам нужно проверить их, используя регулярные выражения:

```
1 import re
2 li = ['9999999999', '999999-999', '999999x9999']
3
4 for val in li:
5     if re.match(r'[8-9]{1}[0-9]{9}', val) and len(val) == 10:
6         print 'yes'
7     else:
8         print 'no'
9
10 Результат:
11 yes
12 no
13 no
```

Задача 7: Разбить строку по нескольким разделителям

Возможное решение:

```

1 import re
2 line = 'asdf fjdk;afed,fjek,asdf,foo' # String has multiple delimiters (";",","," ").
3 result = re.split(r'[;,\s]', line)
4 print result
5
6 Результат:
7 ['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']

```

Также мы можем использовать метод `re.sub()` для замены всех разделителей пробелами:

```

1 import re
2 line = 'asdf fjdk;afed,fjek,asdf,foo'
3 result = re.sub(r'[;,\s]', ' ', line)
4 print result
5
6 Результат:
7 asdf fjdk afed fjek asdf foo

```

Задача 8: Извлечь информацию из html-файла

Допустим, нам надо извлечь информацию из html-файла, заключенную между `<td>` и `</td>`, кроме первого столбца с номером. Также будем считать, что html-код содержится в строке.

Пример html-файла:

```

1 1NoahEmma2LiamOlivia3MasonSophia4JacobIsabella5WilliamAva6EthanMia7MichaelEmily

```

Решение:

```

1 result = re.findall(r'\w+\s(\w+)\s(\w+)', str)
2 print result
3
4 Результат:
5 [('Noah', 'Emma'), ('Liam', 'Olivia'), ('Mason', 'Sophia'), ('Jacob', 'Isabella'), ('William', 'Ava'), ('Ethan', 'Mia'), ('Michael', 'Emily')]

```

Прочитать содержимое html-файла можно с помощью библиотеки `urllib2`:

```
1 import urllib2
2 response = urllib2.urlopen('')
3 html = response.read()
```

Заключение

В этой статье мы узнали, что такое регулярные выражения и как их использовать, на примере библиотеки `re` в Python. Кроме того, мы рассмотрели наиболее часто встречающиеся задачи, в которых их можно применить.

Перевод статьи «[Beginners Tutorial for Regular Expressions in Python](#)».

Для начинающих, Парсинг

♥ This is interesting

◀ 1K

Другим программистам нравится:



[Подборка шпаргалок для программистов](#)



[Разбираемся, как работает встроенная функция zip в Python, и пишем свою реализацию с помощью list comprehension](#)



10 полезных ресурсов для обучения Python



Введение в ООП с примерами на C#. Часть пятая. Всё о модификаторах доступа

10 comments

“

Сломался комп. Ем на кухне.

”

Ответы экспертов



Как лучше действовать, если вы хотите научиться программировать, но не знаете, как встать на истинный путь — готовые инструкции для начинающих от экспертов Tproger
11 июля 2016



С какой платформы лучше начинать мобильную разработку? Обязательно ли сразу выпускаться под все платформы?
1 июня 2016



Как в IT-компаниях смотрят на программистов без диплома при приёме на работу?
11 мая 2016



Какой язык программирования лучше выбрать первым для изучения новичку?
29 марта 2016



Олимпиады по программированию — помогают ли они в реальной работе?
26 февраля 2016

Рубрики

[Видео](#)[Викторины](#)[Еженедельная подборка](#)[Задачи](#)[Инструменты](#)[Интервью](#)[Интересные ссылки](#)[Ищем программиста](#)[Книги](#)[Новости](#)[Ответы экспертов](#)[Переводы](#)[Подборки](#)[Рассказы о своих проектах](#)[События](#)[Спонсорское](#)[Статьи](#)

Уведомления о свежих постах (Chrome, Firefox, Android, Telegram).

Подписаться



Типичный программист

Создано программистами для программистов.

379,739 followers



 Follow on VK

Темы

Android

C#

C++

Git

Google

Hardware

Java

JavaScript

Linux

Microsoft

OpenSource

PHP

Python

StackOverflow

Unity

Windows

Алгоритмы

Безопасность

Безопасный код

Браузеры

Веб-разработка

Головоломки

Для мотивации

Для начинающих

Задачи повышенной сложности

Задачи умеренной сложности

Инструменты

Интернет

История успеха

Лучшая практика

Математика и теория вероятностей

Машинное обучение

Мобильная разработка

Нейронные сети

Низкоуровневое программирование

Обучение программированию

Оптимизация

Потоки и блокировки

Разработка игр

Рекомендуем

Ретро

Собеседование

Тестирование

Язык Си

[О проекте](#) [Реклама](#) admin@tproger.ru



Нашли опечатку? Выделите фрагмент и отправьте нажатием Ctrl+Enter.