Регулярные выражения в Python

Д.Н.Лавров, А. Лапин

2018

Определение

- Регулярное выражение — формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов (*wildcard characters*).
- Для поиска используется строка-образец (*pattern*, «шаблон», «маска»), состоящая из символов и метасимволов и задающая правило поиска.
- Для манипуляций с текстом дополнительно задаётся строка замены, которая также может содержать в себе специальные символы.

Определение простыми словами

- Регулярное выражение — это специального формата строка, используемая для поиска и замены текста в строке или файле.
- Регулярные выражения используются для:
  - поиска в строке;
  - разбиения строки на подстроки;
  - замены части строки.

История

- Истоки регулярных выражений лежат в теории автоматов, теории формальных языков и классификации формальных грамматик по Хомскому.
- Эти области изучают вычислительные модели (автоматы) и способы описания и классификации формальных языков.
- В 1940-х гг. Уоррен Маккалок и Уолтер Питтс описали нейронную систему, используя простой автомат в качестве модели нейрона.
- Математик Стивен Клини позже описал эти модели, используя свою систему математических обозначений, названную «регулярные множества».
- Кен Томпсон встроил их в редактор QED, а затем в редактор **ed** под UNIX. С этого времени регулярные выражения стали широко использоваться и в языках программирования.

Символы

- Регулярные выражения используют два типа символов:
- специальные символы (операторы): *, \, [,], ?,+, {, }, $, ^, |, (,), .,\w, \d, \s, \b, \t, \n, \r и др.
- литералы: все остальные буквы, цифры и символы.

Операторы

Regular expression cheatsheet

Special characters

`\`	escape special characters
`.`	matches any character
`^`	matches beginning of string
`$`	matches end of string
`[5b-d]`	matches any chars '5', 'b', 'c' or 'd'
`[^a-c6]`	matches any char except 'a', 'b', 'c' or '6'
`R|S`	matches either regex `R` or regex `S`
`()`	creates a capture group and indicates precedence

Quantifiers

`*`	0 or more (append `?` for non-greedy)
`+`	1 or more (append `?` for non-greedy)
`?`	0 or 1 (append `?` for non-greedy)
`{m}`	exactly `m` m occurrences
`{m, n}`	from `m` to `n`. `m` defaults to 0, `n` to infinity
`{m, n}?`	from `m` to `n`, as few as possible

Special sequences

`\A`	start of string
`\b`	matches empty string at word boundary (between `\w` and `\W`)
`\B`	matches empty string not at word boundary
`\d`	digit
`\D`	non-digit
`\s`	whitespace: `[ \t\n\r\f\v]`
`\S`	non-whitespace
`\w`	alphanumeric: `[0-9a-zA-Z_]`
`\W`	non-alphanumeric
`\Z`	end of string
`\g<id>`	matches a previously defined group

Special sequences

`(?iLmsux)`	matches empty string, sets re.X flags
`(?:...)`	non-capturing version of regular parentheses
`(?P...)`	matches whatever matched previously named group
`(?P=)`	digit
`(?#...)`	a comment; ignored
`(?=...)`	lookahead assertion: matches without consuming
`(?!...)`	negative lookahead assertion
`(?<=...)`	lookbehind assertion: matches if preceded
`(?<!...)`	negative lookbehind assertion
`(?(id)yes|no)`	match 'yes' if group 'id' matched, else 'no'

Based on tartley's python-regex-cheatsheet.

Подключение пакета

- import re
- from re import *

Основные функции пакета

- **`re.match()`** – сравнение с шаблоном
- **`re.search()`** – поиск шаблона во всей строке
- **`re.findall()`** – поиск всех вхождений шаблона
- **`re.split()`** – разделение строки на подстроки
- **`re.sub()`** – замена, подстановка
- **`re.compile()`** – скомпилировать (подготовить) регулярное выражения для повторного использования

`re.match()`

- **re.match(*pattern*, *string*)**
- Этот метод ищет по заданному шаблону в начале строки.

re.match()

- **Пример**

```
import re

print(re.match(r"ФКН","ФКН Python ФКН"))
print(re.match(r"ФКН","ФКН Python ФКН").start(), \
      re.match(r"ФКН","ФКН Python ФКН").end())

print(re.match(r"Python","ФКН Python ФКН"))
```

- **Вывод**

```
<_sre.SRE_Match object; span=(0, 3), match='ФКН'>
0 3
None
```

- **r –** отключение механизма экранирования («сырая» строка). Иначе вместо r«\n» нужно писать «\\n». Пример ищем «\section» в RegExp это будет r«\\section» или «\\\\section». На слэнге это называют «бэкслэш проклятие».
- Нельзя заканчивать «сырую» строку слэшем «\».

`re.search()`

- **re.search(*pattern*, *string*)**
- Метод **search**() ищет по всей строке, но возвращает только первое найденное совпадение.

re.search()

- **Пример**

  res = re.search(**r'Python'**, **'ФКН Python Python ФКН'**)
  print(res.start(), res.end())

  print(res.span())

  print(res.group(0))
  print(res.group(1))
- **Вывод**

  4 10

  (4, 10)

  Traceback (most recent call last):

  Python

    File “RegExpr.py", line 13, in <module>

      print(res.group(1))

  IndexError: no such group

`re.findall()`

- **re.findall(*pattern*, *string*)**
- Метод возвращает список всех найденных совпадений. У метода **findall**() нет ограничений на поиск в начале или конце строки.

`re.findall()`

- **Пример**

```
res = re.findall(r'ФКН', \
                 'ФКН Python Python ФКН')
print(res)
```

- **Вывод**

```
['ФКН', 'ФКН']
```

`re.split()`

- **re.split(*pattern*, *string*, *[maxsplit=0]*)**
- Этот метод разделяет строку по заданному шаблону.

re.split()

- **Пример**

  res = re.split(**r' '**, **'ФКН Python Python ФКН'**)
  print(res)
  res = re.split(**r'yt'**, **'ФКН Python Python ФКН'**)
  print(res)

  res = re.split(**r'[y,o]'**, **'ФКН Python Python ФКН'**)
  print(res)

- **Вывод**

  ['ФКН', 'Python', 'Python', 'ФКН']

  ['ФКН P', 'hon P', 'hon ФКН']

  ['ФКН P', 'th', 'n P', 'th', 'n ФКН']

`re.sub()`

- **re.sub(*pattern*, *repl*, *string*):**
- Метод ищет шаблон в строке и заменяет его на указанную подстроку.

re.sub()

- **Пример**

```
res = re.sub(r'ИМИТ', 'ФКН', \
                    'ИМИТ Python Python ИМИТ')
print(res)
```

- **Вывод**

ФКН Python Python ФКН

`re.compile()`

- **re.compile(*pattern*, *repl*, *string*):**

  Избавляет от переписывания одного

  и того же выражения, используется

  для того чтобы не повторят многократно компиляцию одного и того же выражения .

re.compile()

- **Пример**

```
pattern = re.compile('Ф.Н')
res = pattern.findall('ФКН Python Python ФФН')
print(res)
res = pattern.findall('На ФКН мы изучаем Python \
на 2 курсе')
print(res)
```

- **Вывод**

```
['ФКН', 'ФФН']
['ФКН']
```

Часто используемые операторы

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 непробельнй 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

Задача 1: Вернуть первое (последнее) слово из строки

Решение:

```
s="Проба пера: Мама мыла раму"
result = re.findall(r'^\w+', s)
print(result)
result = re.findall(r'\w+$', s)
print(result)
```

Вывод:

```
['Проба']
['раму']
```

Задача 2: Вернуть первые два символа каждого слова

- **Решение**:

  s=**'ФКН Python,P,ython ФКН'**
  result = re.findall(**r'\w\w'**, s)
  print(result)
  result = re.findall(**r'\b\w.'**, s)
  print(result)

  result = re.findall(**r'\b\w\w'**, s)
  print(result)
- **Вывод**:

  ['ФК', 'Py', 'th', 'on', 'yt', 'ho', 'ФК']

  ['ФК', 'Py', 'P,', 'yt', 'ФК']

  ['ФК', 'Py', 'yt', 'ФК']

Задача 3: Вернуть список доменов из списка адресов электронной почты

Решение:

```
s='abc.test@gmail.com, xyz@test,in, ‘+\
   'test.first@analyticsvidhya.com, ‘+\
   'first.test@rest.biz'
   res = re.findall(r'@\w+\.(\w+)', s)
   res = re.findall(r'@\w+.(\w+)', s)
```

Результат:

```
   ['com', 'com', 'biz‘]
   ['com', 'in', 'com', 'biz‘]
```

Задача 4: Извлечь дату из строки

Решение:

```
s= ‘Omsk 34-3456 12-05-2017, XYZ 56-4532 11-11-2011,’+\
    'ABC 67-8945 12-01-2009'
   result = re.findall(r'\d{2}-\d{2}-\d{4}', s)
   print(result)
   result = re.findall(r'\d{2}-\d{2}-(\d{4})', s)
   print(result)
   result = re.findall(r'(\d{2}-\d{2})-(\d{4})', s)
   print(result)
```

Результат:

```
['12-05-2017', '11-11-2011', '12-01-2009']
['2017', '2011', '2009']
[('12-05', '2017'), ('11-11', '2011'), ('12-01', '2009')]
```

Задача 5: Разбить строку по нескольким разделителям

Решение

```
line = 'asdf, fjdk;afed,fjek,asdf,foo'
result = re.split(r'[;,\s]', line)
print(result)
result = re.split(r'[;,\s]*', line)
print(result)
result = re.split(r'[;,\s]+', line)
print(result)
```

Вывод:

```
['asdf', '', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
N:\Anaconda3\lib\re.py:203: FutureWarning: split() requires a non-empty pattern match.
return _compile(pattern, flags).split(string, maxsplit)
['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
```

Литература

- http://pythex.org/ - онлайн калькуляьтор
- https://regex101.com/ - онлайн калькуляьтор
- https://docs.python.org/2/library/re.html
- https://tproger.ru/translations/regular-expression-python/
- https://ru.wikipedia.org/wiki/Регулярные_выражения
- https://habrahabr.ru/post/115825/
- https://ru.wikibooks.org/wiki/Регулярные_выражения
- http://snakeproject.ru/rubric/article.php?art=python_reg_exp