Примеры применения регулярных выражений в Python

*Регулярные выражения, также называемые regex, синтаксис или, скорее, язык для поиска, извлечения и работы с определенными текстовыми шаблонами большего текста. Он широко используется в проектах, которые включают проверку текста, NLP (Обработка естественного языка) и интеллектуальную обработку текста.*

**Введение в регулярные выражения**

[Регулярные выражения, также называемые regex](https://pythonru.com/moduli/modul-re-dlja-reguljarnyh-vyrazhenij-v-python), используются практически во всех языках программирования. В python они реализованы в стандартном модуле re.  
Он широко используется в естественной обработке языка, веб-приложениях, требующих проверки ввода текста (например, адреса электронной почты) и почти во всех проектах в области анализа данных, которые включают в себя интеллектуальную обработку текста.

Эта статья разделена на 2 части.

Прежде чем перейти к синтаксису регулярных выражений, для начала вам лучше понять, как работает модуль re.

Итак, сначала вы познакомитесь с 5 основными функциями модуля re, а затем посмотрите, как создавать регулярные выражения в python.  
Узнаете, как построить практически любой текстовый шаблон, который вам, скорее всего, понадобится при работе над проектами, связанными с поиском текста.

**Что такое шаблон регулярного выражения и как его скомпилировать?**

Шаблон регулярного выражения представляет собой специальный язык, используемый для представления общего текста, цифр или символов, извлечения текстов, соответствующих этому шаблону.

Основным примером является \s+.  
Здесь \ s соответствует любому символу пробела. Добавив в конце оператор +, шаблон будет иметь не менее 1 или более пробелов. Этот шаблон будет соответствовать даже символам tab \t.

В конце этой статьи вы найдете больший список шаблонов регулярных выражений. Но прежде чем дойти до этого, давайте посмотрим, как компилировать и работать с регулярными выражениями.

>>> import re

>>> regex = re.compile('\s+')

Вышеупомянутый код импортирует модуль re и компилирует шаблон регулярного выражения, который соответствует хотя бы одному или нескольким символам пробела.

**Как разбить строку, разделенную регулярным выражением?**

>>> text = """100 ИНФ Информатика

213 МАТ Математика

156 АНГ Английский"""

У меня есть три курса в формате “[Номер курса] [Код курса] [Название курса]”. Интервал между словами разный.

Передо мной стоит задача разбить эти три предмета курса на отдельные единицы чисел и слов. Как это сделать?  
Их можно разбить двумя способами:

* Используя метод re.split.
* Вызвав метод split для объекта regex.

# Разделит текст по 1 или более пробелами

>>> re.split('\s+', text)

# или

>>> regex.split(text)

['100', 'ИНФ', 'Информатика', '213', 'МАТ', 'Математика', '156', 'АНГ', 'Английский']

Оба эти метода работают. Но какой же следует использовать на практике?  
Если вы намерены использовать определенный шаблон несколько раз, вам лучше скомпилировать регулярное выражение, а не использовать re.split множество раз.

## Поиск совпадений с использованием findall, search и match

Предположим, вы хотите извлечь все номера курсов, то есть 100, 213 и 156 из приведенного выше текста. Как это сделать?

### Что делает re.findall()?

#найти все номера в тексте

>>> print(text)

100 ИНФ Информатика

213 МАТ Математика

156 АНГ Английский

>>> regex\_num = re.compile('\d+')

>>> regex\_num.findall(text)

['100', '213', '156']

В приведенном выше коде специальный символ \ d является регулярным выражением, которое соответствует любой цифре. В этой статье вы узнаете больше о таких шаблонах.  
Добавление к нему символа + означает наличие по крайней мере 1 числа.

Подобно +, есть символ \*, для которого требуется 0 или более чисел. Это делает наличие цифры не обязательным, чтобы получилось совпадение. Подробнее об этом позже.

В итоге, метод findall извлекает все вхождения 1 или более номеров из текста и возвращает их в список.

### re.search() против re.match()

Как понятно из названия, regex.search() ищет шаблоны в заданном тексте.  
Но, в отличие от findall, который возвращает согласованные части текста в виде списка, regex.search() возвращает конкретный объект соответствия. Он содержит первый и последний индекс первого соответствия шаблону.

Аналогично, regex.match() также возвращает объект соответствия. Но разница в том, что он требует, чтобы шаблон находился в начале самого текста.

>>> # создайте переменную с текстом

>>> text2 = """ИНФ Информатика

213 МАТ Математика 156"""

>>> # скомпилируйте regex и найдите шаблоны

>>> regex\_num = re.compile('\d+')

>>> s = regex\_num.search(text2)

>>> print('Первый индекс: ', s.start())

>>> print('Последний индекс: ', s.end())

>>> print(text2[s.start():s.end()])

Первый индекс: 17

Последний индекс: 20

213

В качестве альтернативы вы можете получить тот же результат, используя метод group() для объекта соответствия.

>>> print(s.group())

205

>>> m = regex\_num.match(text2)

>>> print(m)

None

## Как заменить один текст на другой, используя регулярные выражения?

Для изменения текста, используйте regex.sub().  
Рассмотрим следующую измененную версию текста курсов. Здесь добавлена табуляция после каждого кода курса.

# создайте переменную с текстом

>>> text = """100 ИНФ \t Информатика

213 МАТ \t Математика

156 АНГ \t Английский"""

>>> print(text)

100 ИНФ Информатика

213 МАТ Математика

156 АНГ Английский

Из вышеприведенного текста я хочу удалить все лишние пробелы и записать все слова в одну строку.

Для этого нужно просто использовать regex.sub для замены шаблона \s+ на один пробел .

# заменить один или больше пробелов на 1

>>> regex = re.compile('\s+')

>>> print(regex.sub(' ', text))

или

>>> print(re.sub('\s+', ' ', text))

101 COM Computers 205 MAT Mathematics 189 ENG English

Предположим, вы хотите избавиться от лишних пробелов и выводить записи курса с новой строки. Чтобы это сделать, используйте регулярное выражение, которое пропускает символ новой строки, но учитывает все другие пробелы.

Это можно сделать, используя отрицательное соответствие (?!\n). Шаблон проверяет наличие символа новой [строки, в python](https://pythonru.com/osnovy/stroki-python) это \n, и пропускает его.

# убрать все пробелы кроме символа новой строки

>>> regex = re.compile('((?!\n)\s+)')

>>> print(regex.sub(' ', text))

100 ИНФ Информатика

213 МАТ Математика

156 АНГ Английский

## Группы регулярных выражений

Группы регулярных выражений — функция, позволяющая извлекать нужные объекты соответствия как отдельные элементы.

Предположим, что я хочу извлечь номер курса, код и имя как отдельные элементы. Не имея групп мне придется написать что-то вроде этого.

>>> text = """100 ИНФ Информатика

213 МАТ Математика

156 АНГ Английский"""

# извлечь все номера курсов

>>> re.findall('[0-9]+', text)

# извлечь все коды курсов (для латиницы [A-Z])

>>> re.findall('[А-ЯЁ]{3}', text)

# извлечь все названия курсов

>>> re.findall('[а-яА-ЯёЁ]{4,}', text)

['100', '213', '156']

['ИНФ', 'МАТ', 'АНГ']

['Информатика', 'Математика', 'Английский']

Давайте посмотрим, что получилось.  
Я скомпилировал 3 отдельных регулярных выражения по одному для соответствия номерам курса, коду и названию.  
Для номера курса, шаблон [0-9]+ указывает на соответствие всем числам от 0 до 9. Добавление символа + в конце заставляет найти по крайней мере 1 соответствие цифрам 0-9. Если вы уверены, что номер курса, будет иметь ровно 3 цифры, шаблон мог бы быть [0-9] {3}.

Для кода курса, как вы могли догадаться, [А-ЯЁ]{3} будет совпадать с 3 большими буквами алфавита А-Я подряд (буква “ё” не включена в общий диапазон букв).

Для названий курса, [а-яА-ЯёЁ]{4,} будем искать а-я верхнего и нижнего регистра, предполагая, что имена всех курсов будут иметь как минимум 4 символа.

Можете ли вы догадаться, каков будет шаблон, если максимальный предел символов в названии курса, скажем, 20?  
Теперь мне нужно написать 3 отдельные строки, чтобы разделить предметы. Но есть лучший способ. Группы регулярных выражений.  
Поскольку все записи имеют один и тот же шаблон, вы можете создать единый шаблон для всех записей курса и внести данные, которые хотите извлечь из пары скобок ().

# создайте группы шаблонов текста курса и извлеките их

>>> course\_pattern = '([0-9]+)\s\*([А-ЯЁ]{3})\s\*([а-яА-ЯёЁ]{4,})'

>>> re.findall(course\_pattern, text)

[('100', 'ИНФ', 'Информатика'), ('213', 'МАТ', 'Математика'), ('156', 'АНГ', 'Английский')]

Обратите внимание на шаблон номера курса: [0-9]+, код: [А-ЯЁ]{3} и название: [а-яА-ЯёЁ]{4,} они все помещены в круглую скобку (), для формирования группы.

## Что такое “жадное” соответствие в регулярных выражениях?

По умолчанию, регулярные выражения должны быть жадными. Это означает, что они пытаются извлечь как можно больше, пока соответствуют шаблону, даже если требуется меньше.

Давайте рассмотрим пример фрагмента HTML, где нам необходимо получить тэг HTML.

>>> text = "<body>Пример жадного соответствия регулярных выражений</body>"

>>> re.findall('<.\*>', text)

['<body>Пример жадного соответствия регулярных выражений</body>']

Вместо совпадения до первого появления ‘>’, которое, должно было произойти в конце первого тэга тела, он извлек всю строку. Это по умолчанию “жадное” соответствие, присущее регулярным выражениям.

С другой стороны, ленивое соответствие “берет как можно меньше”. Это можно задать добавлением ? в конец шаблона.

>>> re.findall('<.\*?>', text)

['<body>', '</body>']

Если вы хотите получить только первое совпадение, используйте вместо этого метод поиска search.

re.search('<.\*?>', text).group()

'<body>'

## Наиболее распространенный синтаксис и шаблоны регулярных выражений

Теперь, когда вы знаете как пользоваться модулем re, давайте рассмотрим некоторые обычно используемые шаблоны подстановок.

**Основной синтаксис**

|  |  |
| --- | --- |
| . | Один символ кроме новой строки |
| \. | Просто точка ., обратный слеш \ убирает магию всех специальных символов. |
| \d | Одна цифра |
| \D | Один символ кроме цифры |
| \w | Один буквенный символ, включая цифры |
| \W | Один символ кроме буквы и цифры |
| \s | Один пробельный (включая таб и перенос строки) |
| \S | Один не пробельный символ |
| \b | Границы слова |
| \n | Новая строка |
| \t | Табуляция |

**Модификаторы**

|  |  |
| --- | --- |
| $ | Конец строки |
| ^ | Начало строки |
| ab|cd | Соответствует ab или de. |
| [ab-d] | Один символ: a, b, c, d |
| [^ab-d] | Любой символ, кроме: a, b, c, d |
| () | Извлечение элементов в скобках |
| (a(bc)) | Извлечение элементов в скобках второго уровня |

**Повторы**

|  |  |
| --- | --- |
| [ab]{2} | 2 непрерывных появления a или b |
| [ab]{2,5} | от 2 до 5 непрерывных появления a или b |
| [ab]{2,} | 2 и больше непрерывных появления a или b |
| + | одно или больше |
| \* | 0 или больше |
| ? | 0 или 1 |

## Примеры регулярных выражений

**Любой символ кроме новой строки**

>>> text = 'python.org'

>>> print(re.findall('.', text)) # Любой символ кроме новой строки

['p', 'y', 't', 'h', 'o', 'n', '.', 'o', 'r', 'g']

>>> print(re.findall('...', text))

['pyt', 'hon', '.or']

**Точки в строке**

>>>text = 'python.org'

>>> print(re.findall('\.', text)) # соответствует точке

['.']

>>> print(re.findall('[^\.]', text)) # соответствует всему кроме точки

['p', 'y', 't', 'h', 'o', 'n', 'o', 'r', 'g']

**Любая цифра**

>>> text = '01, Янв 2018'

>>> print(re.findall('\d+', text)) # Любое число (1 и более цифр подряд)

['01', '2018']

**Все, кроме цифры**

>>> text = '01, Янв 2018'

>>> print(re.findall('\D+', text)) # Любая последовательность, кроме цифр

[', Янв ']

**Любая буква или цифра**

>>> text = '01, Янв 2018'

>>> print(re.findall('\w+', text)) # Любой символ(1 или несколько подряд)

['01', 'Янв', '2018']

**Все, кроме букв и цифр**

>>> text = '01, Янв 2018'

>>> print(re.findall('\W+', text)) # Все кроме букв и цифр

[', ', ' ']

**Только буквы**

>>> text = '01, Янв 2018'

>>> print(re.findall('[а-яА-ЯёЁ]+', text)) # Последовательность букв русского алфавита

['Янв']

**Соответствие заданное количество раз**

>>> text = '01, Янв 2018'

>>> print(re.findall('\d{4}', text)) # Любые 4 цифры подряд

['2018']

>>> print(re.findall('\d{2,4}', text))

['01', '2018']

**1 и более вхождений**

>>> print(re.findall(r'Co+l', 'So Cooool')) # 1 и более буква 'o' в строке

['Cooool']

**Любое количество вхождений (0 или более раз)**

>>> print(re.findall(r'Pi\*lani', 'Pilani'))

['Pilani']

**0 или 1 вхождение**

>>> print(re.findall(r'colou?r', 'color'))

['color']

**Граница слова**  
Границы слов \b обычно используются для обнаружения и сопоставления началу или концу слова. То есть, одна сторона является символом слова, а другая сторона является пробелом и наоборот.

Например, регулярное выражение \btoy совпадает с ‘toy’ в ‘toy cat’, но не в ‘tolstoy’. Для того, чтобы ‘toy’ соответствовало ‘tolstoy’, используйте toy\b.  
Можете ли вы придумать регулярное выражение, которое будет соответствовать только первой ‘toy’в ‘play toy broke toys’? (подсказка: \ b с обеих сторон)  
Аналогично, \ B будет соответствовать любому non-boundary( без границ).  
Например, \ Btoy \ B будет соответствовать ‘toy’, окруженной словами с обеих сторон, как в ‘antoynet’.

>>> re.findall(r'\btoy\b', 'play toy broke toys') # соедини toy с ограничениями с обеих сторон

['toy']