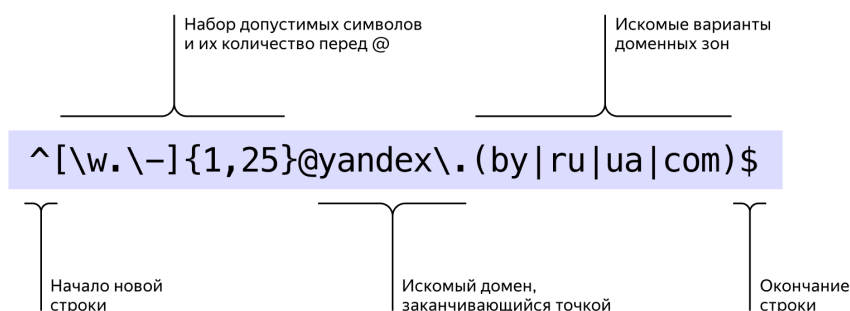


Регулярные выражения | Я.Шпора

Регулярные выражения — это самостоятельный язык, предназначенный для поиска строк, соответствующих заданному шаблону.

Шаблон для поиска — это строка-образец (англ. *pattern*, по-русски её часто называют «шаблоном» или «маской»).

Например, шаблон для поиска строки, содержащей любой адрес электронной почты в доменах yandex.by, yandex.ru, yandex.ua или yandex.com, может быть таким:



- `^` означает начало новой строки.
- `[\w.\-]` — квадратные скобки объединяют набор возможных символов, которые могут находиться в начале строки. В примере маски показано, что среди этих символов могут быть:
 - `\w` — любые словообразующие символы: буквы от а до z и от А до Z, цифры от 0 до 9, знак подчёркивания;
 - `.` — символ «точка»;
 - `\-` — символ «дефис» (экранирован, значит, это символ, который может встретиться в строке).

Эти символы используются чаще всего в первой части (в логине) адреса электронной почты.

- `{1,25}` указывает на то, что строка, состоящая из символов, перечисленных в квадратных скобках перед фигурными скобками, может быть длиной от 1 до 25 знаков.
- Элемент `(...)` объединяет варианты написания доменов, которые могут быть в строке, а разделяющий их символ `|` соответствует оператору «или».
- `$` означает окончание строки.

Специальные символы и примеры их использования:

Символ	Описание	Пример	Результат
<code>^</code>	Признак начала строки	<code>^kittygram</code>	Найдёт все строки, которые начинаются на <code>kittygram</code> , например: <code>kittygram — лучший сервис для котоводов</code>
<code>\$</code>	Признак конца строки	<code>kittygram\$</code>	Найдёт все строки, которые заканчиваются на <code>kittygram</code> , например: <code>лучший сервис для котоводов — это kittygram</code>
<code>.</code>	Точка означает любой символ, кроме перевода строки	<code>Стёпа ..дит</code>	Найдёт строки <code>Стёпа кодит</code> , <code>Стёпа водит</code> , <code>Стёпа ходит</code> и прочие подобные
<code> </code>	Означает ИЛИ	<code>Мурка Снежок</code>	Найдёт как <code>Мурка</code> , так и <code>Снежок</code>
<code>?</code>	Символ слева от <code>?</code> отсутствует или представлен в одном экземпляре	<code>kittygra?m</code>	Найдёт <code>kittygram</code> и <code>kittygrm</code>
<code>*</code>	Символ слева от <code>*</code> может отсутствовать или повторяться сколько угодно раз	<code>kittygra*m</code>	Найдёт <code>kittygram</code> , <code>kittygrm</code> , <code>kittygraam</code> и подобные строки
<code>+</code>	Символ слева от <code>+</code> должен присутствовать хотя бы в одном экземпляре или повторяться сколько угодно раз	<code>kittygra+m</code>	Найдёт <code>kittygram</code> , <code>kittygraam</code> , <code>kittygraaam</code> и подобные строки
<code>\</code>	Символ экранирования или начала метасимвола	<code>dev\project\kittygram</code>	Найдёт текст <code>dev/project/kittygram</code>

Экранирование

Одни и те же символы в шаблоне могут применяться и в качестве метасимволов, и в качестве искомого символа. Например, символ «точка» в

качестве метасимвола обозначает «любой символ», но могут быть ситуации, когда нужно просто найти символ точки в строке.

Чтобы в шаблоне метасимвол отличался от обычного символа — применяется **экранирование**: перед обычным символом ставится обратная косая черта `\`. Таким образом

- символ `.` в шаблоне означает «любой символ»; символ `|` означает «логическое ИЛИ»;
- экранированный символ `\.` в шаблоне означает «символ точки»;
- экранированный символ `\|` означает «символ пайп»;

Метасимволы для групп символов

Помимо специальных символов в регулярных выражениях применяются **метасимволы**: они описывают группы символов в строке. Например:

Мета-символ	Описание	Пример	Результат
<code>\w</code>	Буква, цифра или нижнее подчёркивание	<code>\w+</code>	Соответствует целому слову без пробелов, например <code>Мурка_2010</code>
<code>\W</code>	НЕ буква, цифра или нижнее подчёркивание	<code>\W\w+\W</code>	Найдёт полное слово, которое обрамлено любыми символами, например <code>?Мурка!</code>
<code>\d</code>	Любая цифра	<code>\d+</code>	Соответствует строке из цифр, например <code>128</code> , <code>3</code> или <code>100001</code> . С помощью <code>\d</code> можно, например, описать числа, состоящие из шести цифр: <code>^d{6}\$</code>
<code>\D</code>	Любой символ НЕ цифра	<code>^D+\$</code>	Соответствует любому выражению, где нет цифр, например <code>Мурка</code>

Модуль re

Для работы с регулярными выражениями в Python используется модуль стандартной библиотеки `re`.

```
# Импорт модуля для работы с регулярными выражениями.  
import re
```

Один из самых востребованных методов — `re.search()`, он ищет в строке определённую последовательность символов, задаваемую шаблоном. У него два обязательных параметра:

- `string` — строка, в которой осуществляется поиск;
- `pattern` — шаблон, написанный на языке регулярных выражений, который определяет правила поиска.

Для любой строки можно подобрать шаблон.

Например, такой строке...

```
string = 'Python 3.11 (in development)'.
```

...соответствует такой шаблон (это один из возможных вариантов):

```
pattern = r'Python \d\.\d+ \(.*\).'
```

Сырые строки

Перед строкой стоит буква `r` (от англ. *raw string* — «сырая строка») — она указывает на тип строки и не относится к шаблону.

r-строки отключают экранирование символов, за которое отвечает символ обратного слеша `\`. В результате в такой строке служебные последовательности, например, `\n` или `\t`, становятся обычными символами.

С регулярками удобно работать через r-строки, потому что в них часто используются обратные слешы — либо в последовательностях, либо для экранирования

Регулярное выражение, записанное в сырой строке...

```
pattern = r'Python \d\.\d+ \(.*\).'
```

...будет полностью эквивалентно записи в обычной строке:

```
pattern = 'Python \\d\\.\\d+ \\(.*\\)'
```



В обычной строке первый обратный слеш экранирует второй, и при разборе интерпретатором Python два обратных слеша превращаются в один. Перегружено и нечитаемо, а «читаемость имеет значение».

Использование модуля re

Основные функции:

Функция	Её смысл
<code>re.search(pattern, string)</code>	Найти в строке <code>string</code> первую строчку, подходящую под шаблон <code>pattern</code> ;
<code>re.fullmatch(pattern, string)</code>	Проверить, подходит ли строка <code>string</code> под шаблон <code>pattern</code> ;
<code>re.split(pattern, string, maxsplit=0)</code>	Аналог <code>str.split()</code> , только разделение происходит по подстрокам, подходящим под шаблон <code>pattern</code> ;
<code>re.findall(pattern, string)</code>	Найти в строке <code>string</code> все непересекающиеся шаблоны <code>pattern</code> ;
<code>re.finditer(pattern, string)</code>	Итератор по всем непересекающимся шаблонам <code>pattern</code> в строке <code>string</code> (выдаются <code>match</code> -объекты);
<code>re.sub(pattern, repl, string, count=0)</code>	Заменить в строке <code>string</code> все непересекающиеся шаблоны <code>pattern</code> на <code>repl</code> ;

Пример использования функций:

```
import re

match = re.search(r'\d\d\D\d\d', r'Телефон 123-12-12')
print(match[0] if match else 'Not found')
# -> 23-12

match = re.search(r'\d\d\D\d\d', r'Телефон 1231212')
print(match[0] if match else 'Not found')
# -> Not found

match = re.fullmatch(r'\d\d\D\d\d', r'12-12')
```

```

print('YES' if match else 'NO')
# -> YES
match = re.fullmatch(r'\d\d\D\d\d', r'T. 12-12')
print('YES' if match else 'NO')
# -> NO

print(re.split(r'\W+', 'Где, скажите мне, мои очки?!'))
# -> ['Где', 'скажите', 'мне', 'мои', 'очки', '']

print(re.findall(r'\d\d\.\d\d\.\d{4}',
                 r'Эта строка написана 19.01.2018, а могла бы и 01.09.2017'))
# -> ['19.01.2018', '01.09.2017']

for m in re.finditer(r'\d\d\.\d\d\.\d{4}', r'Эта строка написана :'):
    print('Дата', m[0], 'начинается с позиции', m.start())
# -> Дата 19.01.2018 начинается с позиции 20
# -> Дата 01.09.2017 начинается с позиции 45

print(re.sub(r'\d\d\.\d\d\.\d{4}',
             r'DD.MM.YYYY',
             r'Эта строка написана 19.01.2018, а могла бы и 01.09.2017'))
# -> Эта строка написана DD.MM.YYYY, а могла бы и DD.MM.YYYY

```

Именованные группы

Именованная группа в регулярных выражениях — это механизм, который позволяет задать имя для определенной группы символов в регулярном выражении. Присвоение имени позволяет обращаться к соответствующим частям текста, которые соответствуют этой группе, используя имя, вместо порядкового номера группы.

Именованные группы создаются с помощью синтаксиса `(?P<имя>)`, где `<имя>` — это имя, которое присвоено группе. Например:

```

import re

text = "apple orange banana"

```

```
# Именованная группа.  
pattern = re.compile(r"(?P<fruit>\w+)")  
  
match = pattern.search(text)  
if match:  
    print(match.group("fruit")) # Обращение к группе по имени.
```

Полезные ссылки

[Документация модуля re](#)

[Сайт Regex101](#) для тестирования регулярных выражений. На нём можно проверить работу «регулярок», посмотреть подробную расшифровку шаблонов и получить подсказку по синтаксису.

Я Практикум