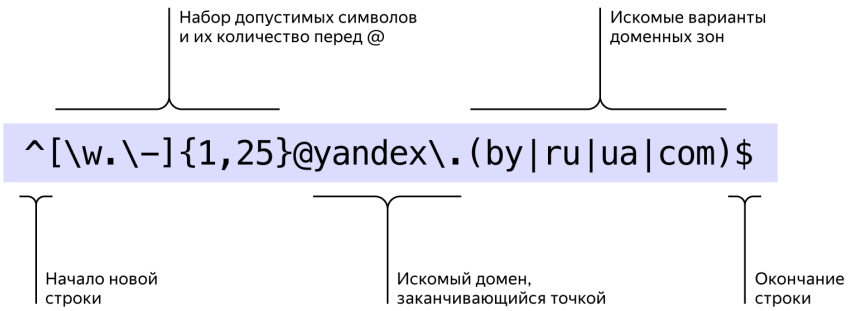
Регулярные выражения | Я.Шпора

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

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

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

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
# -> ['19.01.2018', '01.09.2017']

for m in re.finditer(r'\d\d\.\d\d\.\d{4}', r'Эта строка написана 1
    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
# -> 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.

Практикум