Регулярные выражения в Python: подробное руководство с примерами

Регулярные выражения (regex) — мощный инструмент для поиска и обработки текста в Python. Модуль re предоставляет все необходимые функции для работы с регулярными выражениями. В этой статье рассмотрим основные метасимволы и практические примеры их использования.

Основные метасимволы регулярных выражений

Метасимвол	Описание	Пример	Соответствие
.	Любой символ кроме новой строки	a.c	abc, a1c, a@c
^	Начало строки	^hello	hello world (в начале)
$	Конец строки	world$	hello world (в конце)
*	0 или более повторений	ab*c	ac, abc, abbc
+	1 или более повторений	ab+c	abc, abbc (не ac)
?	0 или 1 повторение	ab?c	ac, abc
{n}	Ровно n повторений	a{3}	aaa
{n,}	n или более повторений	a{2,}	aa, aaa, aaaa
{n,m}	От n до m повторений	a{2,4}	aa, aaa, aaaa
[]	Символьный класс	[abc]	a, b, c
[^]	Отрицание класса	[^abc]	Любой символ кроме a, b, c
|	Альтернатива (ИЛИ)	cat|dog	cat или dog

Метасимвол	Описание	Пример	Соответствие
()	Группировка	(ab)+	ab, abab, abababab
\	Экранирование	\$	Буквальный символ $

Символьные классы

Класс	Описание	Эквивалент
\d	Цифра	[0-9]
\D	Не цифра	[^0-9]
\w	Буква, цифра или подчёркивание	[a-zA-Z0-9_]
\W	Не буква, цифра или подчёркивание	[^a-zA-Z0-9_]
\s	Пробельный символ	[\t\n\r\f\v]
\S	Не пробельный символ	[^ \t\n\r\f\v]

Квантификаторы (жадные и ленивые)

Жадный	Ленивый	Описание
*	*?	0 или более (минимум)
+	+?	1 или более (минимум)
?	??	0 или 1 (минимум)
{n,m}	{n,m}?	От n до m (минимум)

Границы слов

Метасимвол	Описание	Пример
\b	Граница слова	\bcat\b находит cat как отдельное слово
\B	Не граница слова	\Bcat\B находит cat внутри слова

Особые последовательности

Последовательность	Описание
\n	Символ новой строки
\t	Символ табуляции
\r	Символ возврата каретки
\f	Символ перевода страницы
\v	Символ вертикальной табуляции
\0	Нулевой символ
\xhh	Символ с шестнадцатеричным кодом hh

Примеры использования

```
import re

# Поиск email
email_pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'

# Поиск телефона
phone_pattern = r'\+?[1-9]\d{1,14}'

# Поиск даты в формате DD.MM.YYYY
date_pattern = r'\d{2}\.\d{2}\.\d{4}'

# Поиск IP-адреса
ip_pattern = r'\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b'
```

Точка (.) — универсальный символ

Метасимвол точка соответствует любому символу, кроме символа новой строки. Это один из наиболее часто используемых элементов регулярных выражений.

```
import re

text = "cat bat mat rat"
pattern = r".at"  # Находим слова, оканчивающиеся на "at"
matches = re.findall(pattern, text)
print(matches)  # Вывод: ['cat', 'bat', 'mat', 'rat']
```

Якорь начала строки (^)

Символ ^ указывает на начало строки. Он позволяет найти шаблон только в том случае, если он находится в самом начале текста.

```
import re

text = "hello world"
pattern = r"^hello"  # Ищем слово, начинающееся с "hello"
match = re.search(pattern, text)
if match:
    print(match.group())  # Вывод: 'hello'
```

Якорь конца строки ($)

Символ $ соответствует концу строки. Используется для поиска шаблонов в конце текста.

```
import re

text = "hello world"
pattern = r"world$"  # Ищем слово, заканчивающееся на "world"
match = re.search(pattern, text)
if match:
    print(match.group())  # Вывод: 'world'
```

Квантификаторы в регулярных выражениях

Звездочка (*) — ноль или более вхождений

Квантификатор * соответствует нулю или более вхождениям предшествующего элемента.

```
import re

text = "ac abc abbc abbbc"
pattern = r"ab*c"  # Ищем "a", за которым идет ноль или более "b", а затем
"c"
matches = re.findall(pattern, text)
print(matches)  # Вывод: ['ac', 'abc', 'abbc', 'abbbc']
```

Плюс (+) — одно или более вхождений

Квантификатор + требует наличия хотя бы одного вхождения предшествующего элемента.

```
import re

text = "ac abc abbc abbbc"
pattern = r"ab+c"  # Ищем "a", за которым идет один или более "b", а затем "c"
matches = re.findall(pattern, text)
print(matches)  # Вывод: ['abc', 'abbc', 'abbbc']
```

Знак вопроса (?) — ноль или одно вхождение

Квантификатор ? делает предшествующий элемент необязательным.

```
import re

text = "colour color"
pattern = r"colou?r"  # Ищем "colou" с нулевым или одним "u", за которым следует "r"
matches = re.findall(pattern, text)
print(matches)  # Вывод: ['colour', 'color']
```

Практические примеры использования regex

Валидация email-адресов

Поиск и валидация email-адресов — одна из наиболее распространенных задач при работе с регулярными выражениями.

```
import re

text = "Emails: test@example.com, another.test@mail.co.uk"
pattern = r"\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b"
matches = re.findall(pattern, text)
print(matches)  # Вывод: ['test@example.com', 'another.test@mail.co.uk']
```

Извлечение URL из текста

Регулярные выражения помогают находить веб-адреса в тексте различных форматов.

```
import re

text = "Visit my website at https://www.example.com"
pattern = r"https?://(?:www\.)?([a-zA-Z0-9-]+\.[a-zA-Z]{2,})"
matches = re.findall(pattern, text)
print(matches)  # Вывод: ['example.com']
```

Поиск хэштегов в социальных сетях

Для анализа контента социальных сетей часто требуется извлечение хэштегов.

```
import re
```

```
text = "Check out #Python and #DataScience on Twitter!"
pattern = r"#\w+"
matches = re.findall(pattern, text)
print(matches)  # Вывод: ['#Python', '#DataScience']
```

Поиск IP-адресов

Регулярные выражения эффективно находят IP-адреса в логах и конфигурационных файлах.

```
import re

text = "IP addresses: 192.168.1.1 and 10.0.0.1"
pattern = r"\b(?:\d{1,3}\.){3}\d{1,3}\b"
matches = re.findall(pattern, text)
print(matches)  # Вывод: ['192.168.1.1', '10.0.0.1']
```

Парсинг дат в формате DD/MM/YYYY

Извлечение дат из текста с использованием групп захвата.

```
import re

text = "Dates: 20/01/2022 and 31/12/2023"
pattern = r"\b(0[1-9]|[12]\d|3[01])/(0[1-9]|1[0-2])/(\d{4})\b"
matches = re.findall(pattern, text)
print(matches)  # Вывод: [('20', '01', '2022'), ('31', '12', '2023')]
```

Поиск HTML-тегов

Для обработки HTML-разметки используются ленивые квантификаторы.

```
import re

text = "<p>This is a paragraph</p> <a href='https://example.com'>Link</a>"
pattern = r"<.*?>"  # Ленивый поиск всех символов между "<" и ">"
matches = re.findall(pattern, text)
print(matches)  # Вывод: ['<p>', '</p>', "<a href='https://example.com'>",
'</a>']
```

Продвинутые техники работы с regex

Поиск слов с заглавной буквы

Для выделения имен собственных и начала предложений.

```
import re

text = "Hello world, Python is Amazing"
pattern = r"\b[A-Z][a-z]*\b"
matches = re.findall(pattern, text)
print(matches)  # Вывод: ['Hello', 'Python', 'Amazing']
```

Поиск палиндромов

Использование обратных ссылок для поиска слов, которые начинаются и заканчиваются одинаково.

```
import re
```

```
text = "level deed book car door"
pattern = r"\b(\w)\w*\1\b"
matches = re.findall(pattern, text, re.IGNORECASE)
print(matches)  # Находит слова-палиндромы
```

Поиск повторяющихся символов

Выявление слов с двумя или более повторяющимися символами подряд.

```
import re

text = "ssss yyyy aa bb cccc"
pattern = r"\b\w*(\w)\1+\w*\b"
matches = re.findall(pattern, text)
print([match for match in re.finditer(r"\b\w*(\w)\1+\w*\b", text)])
```

Поиск слов определенной длины

Для анализа текста по длине слов используются квантификаторы с фигурными скобками.

```
import re

text = "apple banana orange kiwi"
pattern = r"\b\w{6}\b"  # Ищем слова длиной в 6 символов
matches = re.findall(pattern, text)
print(matches)  # Вывод: ['banana', 'orange']
```

Дополнительные примеры для практики

Поиск номеров телефонов

```
import re

text = "Contact: +7-900-123-45-67 or 8(495)123-45-67"
pattern = r"(\+7|8)[\s\-\(]?(\d{3})[\s\-\)]?(\d{3})[\s\-]?(\d{2})[\s\-
]?(\d{2})"
matches = re.findall(pattern, text)
print(matches)
```

Извлечение чисел из текста

```
import re

text = "Temperature: 25.5°C, Humidity: 60%, Pressure: 1013.25 hPa"
pattern = r"\d+\.?\d*"
matches = re.findall(pattern, text)
print(matches)  # Вывод: ['25.5', '60', '1013.25']
```

Поиск слов в кавычках

```
import re

text = 'He said "Hello world" and then "Goodbye"'
pattern = r'"([^"]*)"'
matches = re.findall(pattern, text)
print(matches)  # Вывод: ['Hello world', 'Goodbye']
```

Регулярные выражения в Python - Полная справочная таблица

Основные методы модуля re

Метод	Описание	Пример использования
re.search(pattern, string, flags=0)	Ищет первое совпадение в строке	re.search(r'\d+', 'abc123def')
re.match(pattern, string, flags=0)	Проверяет совпадение в начале строки	re.match(r'\d+', '123abc')
re.fullmatch(pattern, string, flags=0)	Проверяет полное совпадение всей строки	re.fullmatch(r'\d+', '123')
re.findall(pattern, string, flags=0)	Возвращает все совпадения как список	re.findall(r'\d+', 'a1b2c3')
re.finditer(pattern, string, flags=0)	Возвращает итератор объектов Match	re.finditer(r'\d+', 'a1b2c3')
re.sub(pattern, repl, string, count=0, flags=0)	Заменяет совпадения	re.sub(r'\d+', 'X', 'a1b2c3')
re.subn(pattern, repl, string, count=0, flags=0)	Заменяет и возвращает кортеж (результат, количество замен)	re.subn(r'\d+', 'X', 'a1b2c3')
re.split(pattern, string, maxsplit=0, flags=0)	Разбивает строку по паттерну	re.split(r'\s+', 'a b c')
re.compile(pattern, flags=0)	Компилирует регулярное выражение	re.compile(r'\d+')
re.escape(string)	Экранирует специальные символы	re.escape('a.b*c')

Метод	Описание	Пример использования
re.purge()	Очищает кеш регулярных выражений	re.purge()

Флаги (модификаторы)

Флаг	Константа	Описание	Пример
re.IGNORECASE	re.I	Игнорирует регистр	re.search(r'hello', 'HELLO', re.I)
re.MULTILINE	re.M	^ и $ работают для каждой строки	re.findall(r'^line', text, re.M)
re.DOTALL	re.S	. включает символы новой строки	re.search(r'a.b', 'a\nb', re.S)
re.VERBOSE	re.X	Позволяет использовать комментарии и пробелы	re.compile(r'(?x) \d+ # число')
re.ASCII	re.A	\w, \b, \s и \d работают только с ASCII	re.search(r'\w+', 'café', re.A)
re.LOCALE	re.L	Использует текущую локаль (устарел)	re.search(r'\w+', text, re.L)
re.UNICODE	re.U	Включает Unicode (по умолчанию в Python 3)	re.search(r'\w+', 'café', re.U)

Флаг	Константа	Описание	Пример
re.DEBUG	-	Выводит отладочную информацию	re.compile(r'\d+', re.DEBUG)

Методы объекта Match

Метод	Описание	Пример
match.group(num=0)	Возвращает совпадение группы	match.group(1)
match.groups()	Возвращает все группы как кортеж	match.groups()
match.groupdict()	Возвращает именованные группы как словарь	match.groupdict()
match.start(group=0)	Возвращает начальную позицию	match.start()
match.end(group=0)	Возвращает конечную позицию	match.end()
match.span(group=0)	Возвращает кортеж (начало, конец)	match.span()
match.expand(template)	Расширяет шаблон с группами	match.expand(r'\1-\2')
match.string	Исходная строка	match.string
match.re	Объект регулярного выражения	match.re
match.pos	Позиция начала поиска	match.pos

Метод	Описание	Пример
match.endpos	Позиция конца поиска	match.endpos
match.lastindex	Индекс последней захваченной группы	match.lastindex
match.lastgroup	Имя последней захваченной группы	match.lastgroup

Методы скомпилированного объекта Pattern

Метод	Описание	Пример
pattern.search(string, pos=0, endpos=len(string))	Поиск в строке	compiled.search('text')
pattern.match(string, pos=0, endpos=len(string))	Совпадение с начала	compiled.match('text')
pattern.fullmatch(string, pos=0, endpos=len(string))	Полное совпадение	compiled.fullmatch('text')
pattern.findall(string, pos=0, endpos=len(string))	Все совпадения	compiled.findall('text')
pattern.finditer(string, pos=0, endpos=len(string))	Итератор совпадений	compiled.finditer('text')
pattern.sub(repl, string, count=0)	Замена	compiled.sub('new', 'text')
pattern.subn(repl, string, count=0)	Замена с подсчетом	compiled.subn('new', 'text')
pattern.split(string, maxsplit=0)	Разбиение строки	compiled.split('text')

Метод	Описание	Пример
pattern.pattern	Исходный паттерн	compiled.pattern
pattern.flags	Флаги компиляции	compiled.flags
pattern.groups	Количество групп	compiled.groups
pattern.groupindex	Словарь именованных групп	compiled.groupindex

Специальные последовательности для замены

Последовательность	Описание	Пример
\g<number>	Группа по номеру	re.sub(r'(\d+)', r'\g<1>', text)
\g<name>	Именованная группа	re.sub(r'(?P<num>\d+)', r'\g<num>', text)
\1, \2, ...	Обратная ссылка на группу	re.sub(r'(\w+) (\w+)', r'\2 \1', text)
\g<0>	Вся совпадающая строка	re.sub(r'\d+', r'[\g<0>]', text)
\\	Литеральный обратный слеш	re.sub(r'\d+', r'\\', text)

Исключения

Исключение	Описание
re.error	Ошибка компиляции регулярного выражения

Примеры использования различных методов

```
import re

# Компиляция паттерна
pattern = re.compile(r'(\d{4})-(\d{2})-(\d{2})', re.IGNORECASE)

# Поиск
match = pattern.search('Date: 2023-12-25')
if match:
    print(match.group(0))  # 2023-12-25
    print(match.groups())  # ('2023', '12', '25')

# Замена с использованием групп
text = "Born: 1990-05-15, Died: 2020-10-30"
result = pattern.sub(r'\3/\2/\1', text)
print(result)  # Born: 15/05/1990, Died: 30/10/2020

# Использование флагов
pattern_verbose = re.compile(r'''
    (\d{4})     # год
    -           # разделитель
    (\d{2})     # месяц
    -           # разделитель
    (\d{2})     # день
''', re.VERBOSE)

# Разбиение строки
text = "apple,banana;orange:grape"
fruits = re.split(r'[,;:]', text)
print(fruits)  # ['apple', 'banana', 'orange', 'grape']

# Поиск всех совпадений
numbers = re.findall(r'\d+', 'a1b22c333d4444')
print(numbers)  # ['1', '22', '333', '4444']

# Использование именованных групп
pattern_named = re.compile(r'(?P<year>\d{4})-(?P<month>\d{2})-
(?P<day>\d{2})')
match = pattern_named.search('2023-12-25')
if match:
    print(match.groupdict())  # {'year': '2023', 'month': '12', 'day': '25'}
```