

Синтаксис регулярных выражений

Большинство символов в регулярном выражении представляют сами себя за исключением специальных символов

Раздаточный материал № 131

. ^ (читается «карет») \$ * + ? { } [] \ | ()

Если эти символы должны трактоваться как есть, их следует экранировать с помощью слэша. Некоторые специальные символы теряют свое особое значение, если их разместить внутри квадратных скобок, — в этом случае экранировать их не нужно. Например, чтобы найти точку - \.

Метасимвол . (точка) означает один любой символ, исключая символ новой строки (\n).

Набор символов в квадратных скобках [] называется символьным классом и указывает, что на данном месте в строке может стоять один из перечисленных символов. Если требуется указать символы, которые не входят в указанный набор, то используют символ ^ внутри квадратных скобок.

Раздаточный материал № 132

[09] — соответствует числу 0 или 9;

[0-9] — соответствует любому числу от 0 до 9;

[абв] — соответствует буквам «а», «б» и «в»;

[а-г] — соответствует буквам «а», «б», «в» и «г»;

[а-яё] — соответствует любой букве от «а» до «я»;

[АБВ] — соответствует буквам «А», «Б» и «В»;

[А-ЯЁ] — соответствует любой букве от «А» до «Я»;

[а-яА-ЯёЁ] — соответствует любой русской букве в любом регистре;

[0-9а-яА-ЯёЁа-zA-Z] — любая цифра и любая буква независимо от регистра и языка.

Буква «ё» не входит в диапазон [а-я], а буква «Ё» — в диапазон [А-Я].

[^09] - не цифра 0 или 9;

[^0-9] - не цифра от 0 до 9;

[^а-яА-ЯёЁа-zA-Z] - не буква.

Вместо указания символов можно использовать стандартные классы:

Раздаточный материал № 133

\d — соответствует любой цифре. При указании флага А (ASCII) эквивалентно [0-9];

\w — соответствует любой букве, цифре или символу подчеркивания. При указании флага А (ASCII) эквивалентно [a-zA-Z0-9_];

\s — любой пробельный символ. При указании флага А (ASCII) эквивалентно [\t\n\r\f\v];

\D — не цифра. При указании флага А (ASCII) эквивалентно [^0-9];

\W — не буква, не цифра и не символ подчеркивания. При указании флага А (ASCII) эквивалентно [^a-zA-Z0-9_];

\S — не пробельный символ. При указании флага А (ASCII) эквивалентно [^\t\n\r\f\v].

В Python 3 поддержка Unicode в регулярных выражениях установлена по умолчанию. При этом все классы трактуются гораздо шире. Так, класс \d соответствует не только десятичным цифрам, но и другим цифрам из кодировки Unicode, — например, дробям, класс \w включает не только латинские буквы, но и любые другие, а класс \s охватывает также неразрывные пробелы. Поэтому на практике лучше явно указывать символы внутри квадратных скобок, а не использовать классы.

Раздаточный материал «Флаги»

- ♦ I или IGNORECASE — поиск без учета регистра:

```
import re
p = re.compile(r"^[a-яе]+$", re.I | re.U)
print ("Найдено" if p.search("АБВГДЕЕ") else "Нет")
Найдено
p = re.compile(r"^[a-яе]+$", re.U)
print ("Найдено" if p.search("АБВГДЕЕ") else "Нет")
Нет
```

- ♦ M или MULTILINE — поиск в строке, состоящей из нескольких подстрок, разделенных символом новой строки ("\n"). Символ ^ соответствует привязке к началу каждой подстроки, а символ \$ — позиции перед символом перевода строки;

- ♦ S или DOTALL — метасимвол «точка» по умолчанию соответствует любому символу, кроме символа перевода строки (\n). Символу перевода строки метасимвол «точка» будет соответствовать в присутствии дополнительного модификатора. Символ ^ соответствует привязке к началу всей строки, а символ \$ — привязке к концу всей строки:

```
p = re.compile(r"^. $")
print ("Найдено" if p.search("\n") else "Нет")
Нет
```

```
p = re.compile(r"^. $", re.M)
print ("Найдено" if p.search("\n") else "Нет")
Нет
```

```
p = re.compile(r"^. $", re.S)
print ("Найдено" if p.search("\n") else "Нет")
Найдено
```

- ♦ X или VERBOSE — если флаг указан, то пробелы и символы перевода строки будут проигнорированы. Внутри регулярного выражения можно использовать и комментарии:

```
p = re.compile(r""""^ # Привязка к началу строки
[0-9]+ # Строка должна содержать одну цифру (или более)
$      # Привязка к концу строки
""", re.X | re.S)
print("Найдено" if p.search("1234567890") else "Нет")
Найдено

print("Найдено" if p.search("abcd123") else "Нет")
Нет
```

- ♦ A или ASCII — классы \w, \W, \b, \B, \d, \D, \s и \S будут соответствовать символам в кодировке ASCII (по умолчанию указанные классы соответствуют Unicode-символам);

Флаги и UNICODE, включающие режим соответствия Unicode-символам классов \w, \W, \b, \B, \d, \D, \s и \S, сохранены в Python 3 лишь для совместимости с ранними версиями этого языка и никакого влияния на обработку регулярных выражений не оказывают.

- ♦ L или LOCALE — учитываются настройки текущей локали. Начиная с Python 3.6, могут быть использованы только в том случае, когда регулярное выражение задается в виде значения типов bytes или bytearray.

Позиция внутри строки

Следующие символы позволяют позиционировать регулярное выражение относительно элементов текста: начала и конца строки, границ слова.

Раздаточный материал № 134

`^` Начало текста

`$` Конец текста

`\b` — привязка к началу слова (началом слова считается пробел или любой символ, не являющийся буквой, цифрой или знаком подчеркивания);

`\B` — привязка к позиции, не являющейся началом слова.

Символ `^` теряет свое специальное значение, если он не расположен сразу после открывающей квадратной скобки. Чтобы отменить специальное значение символа `^`, его необходимо указать после всех символов, перед закрывающей квадратной скобкой или сразу после открывающей квадратной скобки. Все специальные символы можно сделать обычными, если перед ними указать символ `\`.

Для создания шаблона (паттерна) регулярного выражения используется функция `compile()`

Раздаточный материал № 135

`<шаблон> = re.compile(<регулярное выражение> [, <флаг>])`

Перед всеми строками, содержащими регулярные выражения, указан модификатор `r`, т.е. используются неформатированные строки. Если модификатор не указать, то все слэши необходимо экранировать.

Раздаточный материал № 136

```
p = re.compile(r'^\w+$')
```

нужно было бы записать так:

```
p = re.compile("^\\w+$")
```

Раздаточный материал № 137

```
import re # Подключаем модуль
d = "29,12.2009" # Вместо точки указана запятая
p = re.compile(r'^[0-3][0-9].[01][0-9].[12][09][0-9][0-9]$')
# Символ "\" не указан перед точкой
if p.search(d):
    print("Дата введена правильно")
else:
    print("Дата введена неправильно")
# Так как точка означает любой символ,
# выведет: Дата введена правильно

p = re.compile(r'^[0-3][0-9]\\.[01][0-9]\\.[12][09][0-9][0-9]$')
# Символ "\" указан перед точкой
if p.search(d):
    print("Дата введена правильно")
else:
    print("Дата введена неправильно")
# Так как перед точкой указан символ
# выведет: Дата введена неправильно

p = re.compile(r'^[0-3][0-9][.][01][0-9][.][12][09][0-9][0-9]$')
# Точка внутри квадратных скобок
if p.search(d):
    print("Дата введена правильно")
else:
    print("Дата введена неправильно")
# Выведет: Дата введена неправильно
```

В этом примере осуществляется привязка к началу и концу строки с помощью следующих метасимволов:

A — привязка к началу строки или подстроки. Она зависит от флагов **M** (или **MULTILINE**) и **S** (или **DOTALL**);

\$ — привязка к концу строки или подстроки. Она зависит от флагов **M** (или **MULTILINE**) и **S** (или **DOTALL**);

\A — привязка к началу строки (не зависит от модификатора);

\Z — привязка к концу строки (не зависит от модификатора).

Если указан флаг **M** (или **MULTILINE**), то поиск производится в строке, состоящей из нескольких подстрок, разделенных символом новой строки (**\n**). В этом случае символ **^** соответствует привязке к началу каждой подстроки, а ствол **\$** — позиции перед стволом перевода строки:

Раздаточный материал № 138

```
import re
p = re.compile(r"^.+$") # Точка не соответствует \n
print(p.findall("str1\nstr2\nstr3")) # Ничего не найдено []

p = re.compile(r"^.+$", re.S) # Теперь точка соответствует \n
print(p.findall("str1\nstr2\nstr3")) # Строка полностью
# соответствует ['str1\nstr2\nstr3']

p = re.compile(r"^.+$", re.M) # Многострочный режим
print(p.findall("str1\nstr2\nstr3")) # Получили каждую подстроку
['str1', 'str2', 'str3']
```

Привязку к началу и концу строки следует использовать, если строка должна полностью соответствовать регулярному выражению. Например, для проверки, содержит ли строка число.

Раздаточный материал № 139

```
import re # Подключаем модуль
p = re.compile(r"^[0-9]+$", re.S)
if p.search("245"):
    print("Число") # Выведет: Число
else:
    print("Не число")

if p.search("Строка245"):
    print("Число")
else:
    print("Не число") # Выведет: Не число
```

Если убрать привязку к началу и концу строки, то любая строка, содержащая хотя бы одну цифру, будет распознана как число.

Раздаточный материал № 140

```
import re # Подключаем модуль
p = re.compile(r"[0-9]+", re.S)
if p.search("Строка245"):
    print("Число") # Выведет: Число
else:
    print("Не число")
```

Кроме того, можно указать привязку только к началу или только к концу строки.

Раздаточный материал № 141

```
# Привязка к началу и концу строки

import re
p = re.compile(r"[0-9]+$", re.S)
if p.search("Строка245"):
    print("Есть число в конце строки")
else:
    print("Нет числа в конце строки")
# Выведет: Есть число в конце строки
```

```
p = re.compile(r"^[0-9]+", re.S)
if p.search("Строка245"):
    print("Есть число в начале строки")
else:
    print("Нет числа в начале строки")
# Выведет: Нет числа в начале строки
```

Раздаточный материал № 142

```
import re

p = re.compile(r"\bpython\b")
print ("Найдено" if p.search ("python") else "Нет")
# выдаст Найдено

print ("Найдено" if p.search("pythonware") else "Нет")
# выдаст Нет

p = re.compile(r"\Bth\B")
print ("Найдено" if p.search("python") else "Нет")
# выдаст Найдено

print ("Найдено" if p.search("this") else "Нет")
# выдаст Нет
```

Метасимвол | позволяет сделать выбор между альтернативными значениями.

Раздаточный материал № 143

```
import re

p = re.compile(r"красн((ая)|(ое))")
print("Найдено" if p.search("красная") else "Нет")
# выдаст Найдено

print("Найдено" if p.search("красное") else "Нет")
# выдаст Найдено

print("Найдено" if p.search("красный") else "Нет")
# выдаст Нет
```

Количество вхождений символа в строку задается с помощью **квантификаторов**:

Раздаточный материал № 144

{n} — n вхождений символа в строку. Например, шаблон r"^[0-9]{2}\$" соответствует двум вхождениям любой цифры;

(n,} — n или более вхождений символа в строку. Например, шаблон r"^[0-9][2,}\$" соответствует двум и более вхождениям любой цифры;

{n,m} — не менее n и не более m вхождений символа в строку. Числа указываются через запятую без пробела. Например, шаблон r"^[0-9]{2,4}\$" соответствует от двух до четырех вхождений любой цифры;

* — ноль или большее число вхождений символа в строку. Эквивалентно комбинации {0,};

+ — одно или большее число вхождений символа в строку. Эквивалентно комбинации {1,};

? — ни одного или одно вхождение символа в строку. Эквивалентно комбинации {0,1}.

Все квантификаторы являются «жадными». При поиске соответствия ищется самая длинная подстрока, соответствующая шаблону, и не учитываются более короткие соответствия.

Раздаточный материал № 145

Получим содержимое всех тегов `` вместе с тегами:

```
import re

s = "<b>Text1</b>Text2<b>Text3</b>"
p = re.compile(r"<b>.*</b>", re.S)
print(p.findall(s))

# выдаст ['<b>Text1</b>Text2<b>Text3</b>']
# ожидалось ['<b>Text1</b>', '<b>Text3</b>']
```

Чтобы ограничить «жадность», необходимо после квантификатора указать символ `?`:

Раздаточный материал № 146

```
p = re.compile(r"<b>.*?</b>", re.S)
print(p.findall(s))

# выдаст ['<b>Text1</b>', '<b>Text3</b>']
```

Если необходимо получить содержимое без тегов, то нужный фрагмент внутри шаблона следует разместить внутри круглых скобок:

Раздаточный материал № 147

```
p = re.compile(r"<b>(.*?)</b>", re.S)
print(p.findall(s))

# выдаст ['Text1', 'Text3']
```

Поиск первого совпадения с шаблоном

Для поиска первого совпадения с шаблоном предназначены следующие функции и методы:

match() — проверяет соответствие с началом строки. Если соответствие найдено, возвращается объект `Match`, в противном случае — значение `None`. Формат метода:

Раздаточный материал № 148

`match(<Строка>[, <Начальная позиция>[, <Конечная позиция>]])`

```
import re

p = re.compile(r"[0-9]+")
print("Найдено" if p.match("str123") else "Нет")
# выдаст Нет

print("Найдено" if p.match("str123", 3) else "Нет")
# выдаст Найдено

print("Найдено" if p.match("123str") else "Нет")
# выдаст Найдено
```

Вместо метода `match()` можно воспользоваться функцией `match()`. Формат функции:

Раздаточный материал № 149

`re.match(<Шаблон>, <Строка>[, <Модификатор>])`

```
p = r"[0-9]+"
print("Найдено" if re.match(p, "str123") else "Нет")
# выдаст Нет
```

В параметре `<шаблон>` указывается строка с регулярным выражением или скомпилированное регулярное выражение. В параметре «модификатор» можно указать

флаги, используемые в функции `compiled`. Если соответствие найдено, то возвращается объект `Match`, в противном случае — значение `None`:

`search()` — проверяет соответствие с любой частью строки. Если соответствие найдено, возвращается объект `Match`, в противном случае — значение `None`. Формат метода:

Раздаточный материал № 150

`search(<Строка>[, <Начальная позиция>[, <Конечная позиция>]])`

```
p = re.compile(r"[0-9]+")
print ("Найдено" if p.search("str123") else "Нет")
# выдаст Найдено

print ("Найдено" if p.search("123str") else "Нет")
# выдаст Найдено

print ("Найдено" if p.search("123str", 3) else "Нет")
# выдаст Нет
```

Вместо метода `search()` можно воспользоваться функцией `search()`. Формат функции:

Раздаточный материал № 151

`re.search(<Шаблон>, <Строка>[, <модификатор>])`

```
p = r"[0-9]+"
print("Найдено" if re.search(p, "str123") else "Нет")
# выдаст Найдено
```

В параметре `<шаблон>` указывается строка с регулярным выражением или скомпилированное регулярное выражение. В параметре `<модификатор>` можно указать флаги, используемые в функции `compile()`. Если соответствие найдено, возвращается объект `Match`, в противном случае — значение `None`.

`fullmatch()` — выполняет проверку, соответствует ли переданная строка регулярному выражению целиком. Если соответствие найдено, то возвращается объект `Match`, в противном случае — значение `None`. Формат метода:

Раздаточный материал № 152

`fullmatch(<Строка>[, <Начальная позиция>[, <Конечная позиция>]])`

```
p = re.compile("[Pp]ython")
print("Найдено" if p.fullmatch("Python") else "Нет")
# выдаст Найдено

print("Найдено" if p.fullmatch("py") else "Нет")
# выдаст Нет

print("Найдено" if p.fullmatch("PythonWare") else "Нет")
# выдаст Нет

print("Найдено" if p.fullmatch("PythonWare", 0, 6) else "Нет")
# выдаст Найдено
```

Вместо метода `fullmatch()` можно воспользоваться функцией `fullmatch()`. Формат функции:

Раздаточный материал № 153

`re.fullmatch(<Шаблон>, <Строка>[, <Модификатор>])`

В параметре `<Шаблон>` указывается строка с регулярным выражением или скомпилированное регулярное выражение. В параметре `<Модификатор>` можно указать флаги, используемые в функции `compile()`. Если строка полностью совпадает с шаблоном, возвращается объект `Match`, в противном случае — значение `None`.

Поиск всех совпадений с шаблоном

Для поиска всех совпадений с шаблоном предназначено несколько функций и методов.

Метод **findall()** ищет все совпадения с шаблоном. Если соответствия найдены, возвращается список с фрагментами, в противном случае возвращается пустой список. Если внутри шаблона есть более одной группы, то каждый элемент списка будет кортежем, а не строкой. Формат метода:

Раздаточный материал № 154

`findall(<Строка>[, <Начальная позиция>[, <Конечная позиция>]])`

```
import re

p = re.compile(r"[0-9]+")
print(p.findall("2007, 2008, 2009, 2010, 2011"))
# выдаст ['2007', '2008', '2009', '2010', '2011']

p = re.compile(r"[a-z]+")
print(p.findall("2007, 2008, 2009, 2010, 2011"))
# выдаст []
```

Вместо метода `findall()` можно воспользоваться функцией `findall()`. Формат функции:

Раздаточный материал № 155

`re.findall(<Шаблон>, <Строка>[, <Модификатор>])`

В параметре <Шаблон> указывается строка с регулярным выражением или скомпилированное регулярное выражение. В параметре <Модификатор> можно указать флаги,

Метод **finditer()** аналогичен методу `findall()`, но возвращает итератор, а не список. На каждой итерации цикла возвращается объект `Match`. Формат метода:

Раздаточный материал № 156

`finditer(<Строка>[, <Начальная позиция>[, <Конечная позиция>]])`

В параметре «флаг» могут быть указаны флаги (или их комбинация через оператор `|`)

Замена в строке

Метод `sub()` ищет все совпадения с шаблоном и заменяет их указанным значением. Если совпадения не найдены, возвращается исходная строка. Метод имеет следующий формат:

Раздаточный материал № 157

`sub(<Новый фрагмент или ссылка на функцию>, <Строка для замены>[, <Максимальное количество замен>])`

```
import re

p = "Это самый сложный урок"
print(re.sub("сложный", "не сложный", p))
# выдаст Это самый не сложный урок
```


Метод `subn()` аналогичен методу `sub()`, но возвращает не строку, а кортеж из двух элементов: измененной строки и количества произведенных замен. Метод имеет следующий формат:

Раздаточный материал № 158

`subn(<Новый фрагмент или ссылка на функцию>, <Строка для замены>
[, <Максимальное количество замен>])`

```
#Заменим все числа в строке на 0:  
p = re.compile(r"[0-9]+")  
print(p.subn("0", "2008, 2009, 2010, 2011"))  
# выдаст ('0, 0, 0, 0', 4)
```

Разбиение строки

Метод `split()` разбивает строку по шаблону и возвращает список подстрок. Его формат:

Раздаточный материал № 159

`split(<Исходная строка>[, <Лимит>])`

Если во втором параметре задано число, то в списке окажется указанное количество подстрок. Если подстрок больше указанного количества, то список будет содержать еще один элемент — с остатком строки:

Раздаточный материал № 160

Содержимое файла `for_split.txt`

```
Этот файл  
создан для демонстрации;  
работы функции split. В результате должен  
получиться; список
```

```
Import re  
  
p = re.compile(r'[\n;,]+' )  
with open('for_split.txt', 'r', encoding='utf-8') as file:  
    text = file.read()  
    reg_name = re.split(p, text)  
print(reg_name)  
  
# выдаст ['Этот файл', 'создан для демонстрации', 'работы функции split. В  
результате должен', 'получиться', ' список']
```

Использование оператора «with»

В Python встроенный инструмент, который помогает упростить чтение и редактирование файлов. Оператор `with` создает диспетчер контекста в Пайтоне, который автоматически закрывает файл по окончании работы в нем.

Т.е. можно выполнять все стандартные операции ввода\вывода, в привычном порядке в пределах блока кода. После ухода из блока кода, файловый дескриптор закроет его, и его уже нельзя будет использовать.