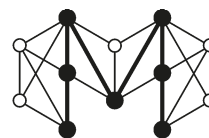


Программирование — 2026.



Введение в регулярные выражения в Python. Модуль `re`

1. Зачем нужны регулярные выражения

Мы уже умеем искать в строке **конкретный** фрагмент: `in`, `find()`, `replace()`.

А как ответить на вопросы:

- состоит ли строка только из цифр и является ли записью целого числа со знаком?
- есть ли в тексте **какое-нибудь** число, и где оно?
- как заменить **любую** последовательность пробелов одним пробелом?

Мы ищем не конкретную строку, а строку **определённого вида**. Описание вида строки называется **шаблоном**, или **регулярным выражением**.

Регулярное выражение — это строка, задающая **множество** строк, элементы которого обладают общей структурой (например, соответствуют шаблону номера телефона, даты рождения, адреса электронной почты и т.п.).

Пример 1

```
In [1]: text = "Заказ 145 оплачен, доставка через 3 дня"

print("145" in text)      # ищем конкретную строку — легко
print(text.find("145"))   # и её позицию

# А как найти ЛЮБОЕ число, не зная его заранее?
# Пришлось бы писать цикл по символам и вручную проверять isdigit()...

True
6
```

Пример 2

Задача. Пользователь вводит дату. Нужно проверить, что она записана в формате ДД.ММ.ГГГГ.

```
In [1]: # правильно, но громоздко
def is_date(s):
    if len(s) != 10:
        return False
    if s[2] != '.' or s[5] != '.':
        return False
    if not s[0:2].isdigit():
        return False
    if not s[3:5].isdigit():
        return False
    if not s[6:10].isdigit():
        return False
    return True

print(is_date('17.10.2026'), is_date('1.2.3'))
```

True False

С "регулярками" — одна строка описания **шаблона**:

```
In [2]: import re

def is_date(s):
    if re.fullmatch(r'\d{2}\.\d{2}\.\d{4}', s):
        return True
    return False

print(is_date('17.10.2026'), is_date('1.2.3'))
```

True False

2. Первый шаблон и модуль `re`

Модуль `re` входит в стандартную библиотеку — устанавливать ничего не нужно.

В этой лекции рассмотрим **три функции**:

Задача	Функция	Результат
Проверить, что вся строка соответствует шаблону	<code>re.fullmatch(шаблон, строка)</code>	объект совпадения или <code>None</code>
Найти первый подходящий фрагмент	<code>re.search(шаблон, строка)</code>	объект совпадения или <code>None</code>
Заменить все подходящие фрагменты	<code>re.sub(шаблон, замена, строка)</code>	новая строка

Объект совпадения в условии `if` ведёт себя как `True`, а `None` — как `False`.

```
In [3]: import re

print(re.fullmatch("кот", "кот"))
print(re.fullmatch("кот", "котик"))
```

```
<re.Match object; span=(0, 3), match='кот'>
None
```

```
In [4]: word = "кот"
if re.fullmatch("кот", word):
    print("Совпадает")
else:
    print("Не совпадает")
```

Совпадает

Пока шаблон "кот" ничем не лучше сравнения `word == "кот"`. Сила шаблонов — в **специальных символах**.

3. Из чего составляется шаблон

3.1. Обычные символы и наборы

Обычный символ (буква, цифра) означает сам себя. Квадратные скобки задают **один** символ из набора.

Запись	Значение	Пример подходящих строк
<code>abc</code>	ровно строка «abc»	<code>abc</code>
<code>[abc]</code>	один символ: <code>a</code> , <code>b</code> или <code>c</code>	<code>a</code> , <code>c</code>
<code>[0-9]</code>	одна цифра (диапазон)	<code>0</code> , <code>7</code>
<code>[a-z]</code>	одна строчная латинская буква	<code>q</code>
<code>[a-яё]</code>	одна строчная русская буква	<code>ж</code> , <code>ё</code>
<code>[a-zA-Z]</code>	одна латинская буква любого регистра	<code>G</code>
<code>[^0-9]</code>	один символ, не являющийся цифрой	<code>x</code> , <code>-</code> , <code>,</code>
<code>.</code>	любой один символ (кроме перевода строки)	<code>a</code> , <code>5</code> , <code>%</code>

⚠ Буква `ё` не входит в диапазон `а-я` — её добавляют отдельно.

```
In [5]: print(re.fullmatch("[кр]от", "кот"))
print(re.fullmatch("[кр]от", "пот"))
print(re.fullmatch("[кр]от", "лот"))
print(re.fullmatch("к.т", "кит"))
print(re.fullmatch("[^0-9]", "a"), re.fullmatch("[^0-9]", "5"))
```

```
<re.Match object; span=(0, 3), match='кот'>
<re.Match object; span=(0, 3), match='пот'>
None
<re.Match object; span=(0, 3), match='кит'>
<re.Match object; span=(0, 1), match='a'> None
```

3.2. Классы символов

Для часто встречающихся наборов есть краткие обозначения:

Запись	Значение	Эквивалент (приблизительно)
<code>\d</code>	цифра	<code>[0-9]</code>
<code>\D</code>	не цифра	<code>[^0-9]</code>
<code>\s</code>	пробельный символ (пробел, <code>\t</code> , <code>\n</code>)	<code>[\t\n]</code>
<code>\S</code>	не пробельный символ	
<code>\w</code>	буква (в т.ч. русская), цифра, <code>_</code>	
<code>\W</code>	инверсия правил для <code>\w</code>	
<code>\b</code>	граница слова	

Правило: **заглавная буква — отрицание** строчной.

3.3. Напоминание: сырые строки `r"..."` и экранирование

Обратная косая черта `\` это символ экранирования и Python должен его обрабатывать. Чтобы он этого не делал, шаблоны записывают как **сырые строки**: `r"\d"`.

Если нужен сам специальный символ (точка, скобка, плюс...), его **экранируют** обратной чертой:

Запись	Значение
<code>\.</code>	символ точки
<code>\+</code> , <code>*</code> , <code>\?</code>	символы <code>+</code> , <code>*</code> , <code>?</code>
<code>\(</code> , <code>\)</code> , <code>\[</code> , <code>\]</code>	скобки
<code>\\</code>	обратная косая черта

Внутри `[...]` большинство спецсимволов теряет особый смысл: `[.+]` — это точка или плюс.

```
In [6]: print(re.fullmatch(r"\d\d", "42"))
print(re.fullmatch(r"\d\d", "4a"))
print(re.fullmatch(r"3.14", "3x14"))    # точка — любой символ!
print(re.fullmatch(r"3\.14", "3x14"))    # экранированная точка
print(re.fullmatch(r"3\..14", "3.14"))

<re.Match object; span=(0, 2), match='42'>
None
<re.Match object; span=(0, 4), match='3x14'>
None
<re.Match object; span=(0, 4), match='3.14'>
```

 **Вопрос аудитории.** Какие из строк подходят под шаблон `r"[A-Z]\d\d"` ?

`"A12"` , `"a12"` , `"B7"` , `"Z00"` , `"Q123"`

```
In [7]: # проверим
for s in "A12", "a12", "B7", "Z00", "Q123":
    print(s, bool(re.fullmatch(r"[A-Z]\d\d", s)))
```

```
A12 True
a12 False
B7 False
Z00 True
Q123 False
```

4. Повторения и варианты

4.1. Квантификаторы

Квантификатор ставится **после** символа (набора, класса, группы) и указывает, сколько раз он повторяется.

Запись	Число повторений	Пример	Подходит
<code>?</code>	0 или 1	<code>-?\d</code>	<code>5</code> , <code>-5</code>
<code>*</code>	0 и более	<code>\d*</code>	<code>`, 7</code> , <code>123</code>
<code>+</code>	1 и более	<code>\d+</code>	<code>7</code> , <code>123</code>
<code>{n}</code>	ровно n	<code>\d{3}</code>	<code>123</code>
<code>{m,n}</code>	от m до n	<code>\d{2,4}</code>	<code>12</code> , <code>1234</code>
<code>{m,}</code>	не менее m	<code>\d{2,}</code>	<code>12</code> , <code>12345</code>

 Квантификатор относится **только к предыдущему элементу**: `ab+` — это `a` , затем одна или более `b` .

Вспомогательная функция `check`

```
In [4]: def check(pattern, strings):
        """Печатает, какие строки целиком соответствуют шаблону."""
        for s in strings:
            print(f"{s!r:20}", "✓" if re.fullmatch(pattern, s) else "X")
```

```
In [5]: # пояснение 1
s = '123'
print(s)
print(repr(s))
print(f'{s!r}')
```

```
123
'123'
'123'
```

```
In [6]: # пояснение 2 - тернарный оператор
x = 1
x = 3 if x>0 else -3
x
```

Out[6]: 3

Построим **шаблон целого числа** постепенно.

```
In [7]: tests = "7", "42", "-42", "+5", "", "4-2", "--3"

print(r"Шаг 1: r'\d+'")
check(r"\d+", tests)
```

```
Шаг 1: r'\d+'
'7'          ✓
'42'         ✓
'-42'        ✗
'+5'         ✗
''           ✗
'4-2'        ✗
'--3'        ✗
```

```
In [8]: print(r"Шаг 2: r'[-?]\d+'")
check(r"[-?]\d+", tests)
```

```
Шаг 2: r'[-?]\d+'
'7'          ✓
'42'         ✓
'-42'        ✓
'+5'         ✗
''           ✗
'4-2'        ✗
'--3'        ✗
```

```
In [9]: print(r"Шаг 3: r'[-+?]\d+'"); check(r"[-+?]\d+", tests)
```

```
Шаг 3: r'[-+?]\d+'
'7'          ✓
'42'         ✓
'-42'        ✓
'+5'         ✓
''           ✗
'4-2'        ✗
'--3'        ✗
```

4.2. Скобки и варианты

Запись	Значение	Пример	Подходит
(...)	объединить часть шаблона в единое целое	(ab)+	ab , abab
A B	вариант: A или B	кот пёс	кот , пёс
(A B)	вариант внутри большего шаблона	(ко ки)т	кот , кит

| имеет самый низкий приоритет: `ab|cd` означает « `ab` или `cd` », а не « `a` , затем `b` или `c` , затем `d` ».

```
In [10]: check(r"(ha)+", ("ha", "hahaha", "hah", ""))
print()
check(r"(кот|пёс)ик", ("котик", "пёсик", "кот", "кит"))
```

'ha'	✓
'hahaha'	✓
'hah'	✗
''	✗

'котик'	✓
'пёсик'	✓
'кот'	✗
'кит'	✗

 **Задача.** Опишите шаблон десятичного числа: необязательный знак, целая часть, необязательная дробная часть через точку.

Подходят: 3, -3.14, +0.5. Не подходят: 3., .5, 3,14.

```
In [11]: number = r"[+-]?\d+(\.\d+)?"
check(number, ("3", "-3.14", "+0.5", "3.", ".5", "3,14", "1.2.3"))
```

'3'	✓
'-3.14'	✓
'+0.5'	✓
'3.'	✗
'.5'	✗
'3,14'	✗
'1.2.3'	✗

Разбор:

- `[+-]?` — знак, может отсутствовать;
- `\d+` — хотя бы одна цифра;
- `(\.\d+)?` — группа «точка и цифры» целиком необязательна.

5. Поиск фрагмента: `re.search()`

`re.fullmatch()` требует совпадения **всей** строки. `re.search()` ищет **первый** подходящий фрагмент в любом месте.

Методы объекта совпадения:

Метод	Результат
<code>m.group()</code>	найденный текст
<code>m.start()</code>	индекс начала фрагмента
<code>m.end()</code>	индекс за последним символом фрагмента

Значит, `text[m.start():m.end()] == m.group()`.

```
In [16]: text = "Заказ 145 оплачен, доставка через 3 дня"

print(re.fullmatch(r"\d+", text))  # вся строка — не число
m = re.search(r"\d+", text)        # но число в ней есть
print(m)
print(m.group(), m.start(), m.end())
print(text[m.start():m.end()])
```

```
None
<re.Match object; span=(6, 9), match='145'>
145 6 9
145
```

⚠ Если ничего не найдено, результат — `None`, и вызов `m.group()` приведёт к ошибке. Всегда сначала проверяйте результат.

```
In [17]: text = "В этой строке нет цифр"
m = re.search(r"\d+", text)
if m:
    print("Найдено:", m.group())
else:
    print("Чисел нет")
```

```
Чисел нет
```

ВАЖНО: `search` находит **первое** совпадение и захватывает **как можно больше** символов.

```
In [12]: m = re.search(r"\d+", "abc 12345 и 678")
print(m.group())  # не "1", а "12345", и не "678"
```

```
12345
```

■ **Задача.** Найти в строке первое время в формате `ЧЧ:ММ` и вывести его отдельно часы и минуты.

```
In [15]: def first_time(text):
    m = re.search(r"\d\d:\d\d", text)
    if m is None:
        return "Время не найдено"
    t = m.group()
    return f"Часы: {int(t[:2])}, минуты: {int(t[3:])}"

print(first_time("Лекция начнётся в 09:45 в аудитории 211"))
print(first_time("Лекция завтра"))
```

```
Часы: 9, минуты: 45
Время не найдено
```

Важно: шаблон `\d\d:\d\d` пропустит и строку `99:99`, не имеющую смысла время.

Регулярное выражение проверяет **форму** строки; проверку смысла (часы < 24 и тд) часто удобнее делать обычным кодом.

6. Замена: `re.sub()`

`re.sub(шаблон, замена, строка)` возвращает **новую** строку, в которой **все** непересекающиеся совпадения заменены. Исходная строка не меняется (строки неизменяемы).

```
In [20]: text = "Телефон 89001234567, код 4512"
print(re.sub(r"\d", "*", text))      # каждая цифра
print(re.sub(r"\d+", "#", text))     # каждое число целиком
print(text)                          # исходная строка не изменилась
```

```
Телефон *****, код ****
Телефон #, код #
Телефон 89001234567, код 4512
```

```
In [21]: messy = "Слишком много пробелов здесь"
print(re.sub(r" +", " ", messy))
print(re.sub(r"\s+", " ", "строка\tc\n разными пробелами"))
```

```
Слишком много пробелов здесь
строка с разными пробелами
```

Итог: три операции

Вопрос	Функция
Вся строка имеет нужный вид?	<code>re.fullmatch()</code>
Есть ли в строке такой фрагмент и где он?	<code>re.search()</code>
Заменить все такие фрагменты	<code>re.sub()</code>

7. Как проверять шаблон. Типичные ошибки

Шаблон — это описание множества строк. Проверять его нужно на двух наборах:

1. строки, которые **должны** подходить;
2. строки, которые **не должны** подходить — особенно граничные случаи: пустая строка, лишний символ в начале/конце, минимальная и максимальная длина.

Ошибка	Пример	Что не так
<code>search</code> вместо <code>fullmatch</code> при проверке	<code>re.search(r"\d+", "ab12")</code>	находит фрагмент, но строка — не число
Неэкранированная точка	<code>r"3.14"</code>	подходит <code>3x14</code>
<code>*</code> вместо <code>+</code>	<code>r"\d*"</code>	подходит пустая строка
Квантификатор не к тому элементу	<code>r"ab+"</code> вместо <code>r"(ab)+"</code>	повторяется только <code>b</code>
Обычная строка вместо сырой	<code>"\b"</code>	Python превращает <code>\b</code> в символ <code>backspace</code>

8. Решение задач

Задача 1. Прочитать шаблон

Опишите словами, какие строки подходят под каждый шаблон, и приведите по одному подходящему и неподходящему примеру.

1. `r"\d{6}"`
2. `r"[A-Z][a-z]+"`
3. `r"(\+7|8)\d{10}"`
4. `r"w+\.py"`

```
In [22]: check(r"\d{6}", ("344090", "34409", "3440900")) # почтовый индекс
print(); check(r"[A-Z][a-z]+", ("Python", "python", "P", "PyThon")) # слово с
print(); check(r"(\+7|8)\d{10}", ("79001234567", "89001234567", "79001234567"))
print(); check(r"w+\.py", ["main.py", "my_lab2.py", "main.рус", ".py"]) # файл
```

'344090'	✓
'34409'	✗
'3440900'	✗
'Python'	✓
'python'	✗
'P'	✗
'PyThon'	✗
'+79001234567'	✓
'89001234567'	✓
'79001234567'	✗
'main.py'	✓
'my_lab2.py'	✓
'main.рус'	✗
' .py'	✗

Задача 2. Номер группы

Номер учебной группы имеет вид: **число курса** (1–6), **точка**, **номер группы** из одной или двух цифр. Например: 1.3 , 4.12 .

Составьте шаблон и проверьте его на строках:

- должны подходить: 1.3 , 4.12 , 6.1
- не должны: 7.1 , 1.123 , 13 , 1,3 , 0.5 , 1.

```
In [16]: pattern = r"[1-6]\.\d{1,2}"

good = "1.3", "4.12", "6.1"
bad = "7.1", "1.123", "13", "1,3", "0.5", "1."
print("Должны подходить:"); check(pattern, good)
print("Не должны подходить:"); check(pattern, bad)
```

Должны подходить:	
'1.3'	✓
'4.12'	✓
'6.1'	✓
Не должны подходить:	
'7.1'	✗
'1.123'	✗
'13'	✗
'1,3'	✗
'0.5'	✗
'1.'	✗

Задача 3. Идентификатор Python

Напишите функцию `is_identifier(s)`, которая возвращает `True`, если `s` — допустимое имя переменной (латинские буквы, цифры, `_`; не начинается с цифры), иначе `False`.

Проверка: `x`, `_tmp`, `value2`, `MAX_N` → `True`; `2x`, `my-var`, `'`, `a b` → `False`.

```
'x'      True
'_tmp'   True
'value2'  True
'MAX_N'   True
'2x'      False
'my-var'  False
''        False
'a b'     False
```

```
"Оплата картой 1234 5678 9012 3456 прошла" → "Оплата картой
**** * прошла"
"Карта 1234567890123456" → "Карта **** *
**** *"
```

```
In [21]: # Решение зависит от трактовки условия.
# 1) Если комбинация вида 12345678 90123456 недопустима
def mask_card1(text):
    return re.sub(r'\d{16}|(\d{4} ){3}\d{4}',
        '**** *', text)
# 2) "возможно" может означать, что где-то есть, где-то нет
def mask_card2(text):
    return re.sub(r"(\d{4} ?){3}\d{4}", "**** *", text)

print(mask_card1("Оплата картой 1234 5678 9012 3456 прошла"))
print(mask_card1("Карта 1234567890123456"))
print(mask_card1("Клиент ввел 12345678 90123456"))
print(mask_card1("Номер заказа 42 0101 0234 1234 98"))

print(mask_card2("Оплата картой 1234 5678 9012 3456 прошла"))
print(mask_card2("Карта 1234567890123456"))
print(mask_card2("Клиент ввел 12345678 90123456"))
print(mask_card2("Номер заказа 42 0101 0234 1234 98"))
```

```
Оплата картой **** * прошла
Карта **** *
Клиент ввел 12345678 90123456
Номер заказа 42 0101 0234 1234 98
Оплата картой **** * прошла
Карта **** *
Клиент ввел **** *
Номер заказа 42 0101 0234 1234 98
```

Задача 6. Нормализация текста

Привести строку к аккуратному виду:

заменить любые последовательности пробельных символов одним пробелом,
убрать пробелы перед знаками препинания , . ! ? .

```
In [22]: def normalize(text):
    text = re.sub(r"\s+", " ", text)          # 1) сжать пробелы
    text = re.sub(r"([.,!?!])", r"\1", text)  # 2) убрать пробел перед знаком
    return text.strip()                       # убрать начальные и конечные пробелы

print(normalize(" Привет , мир !\n\nКак дела ? "))
```

```
Привет, мир! Как дела?
```

Внимание: во второй замене используется `\1` — ссылка на текст, найденный в скобках.

Это «заглядывание вперёд» в тему групп, которая будет подробно разобрана позже.

Альтернатива без групп — несколько вызовов `re.sub(r" ,", ", ", text)` и т.д.

9. Итоги

Шаблон задаёт множество строк. Он строится из:

Элемент	Примеры
обычные символы	<code>a</code> , <code>7</code> , <code>:</code>
наборы	<code>[abc]</code> , <code>[0-9]</code> , <code>[^a-z]</code>
классы	<code>.</code> , <code>\d</code> , <code>\w</code> , <code>\s</code> , <code>\D</code> , <code>\W</code> , <code>\S</code>
квантификаторы	<code>?</code> , <code>*</code> , <code>+</code> , <code>{n}</code> , <code>{m,n}</code>
группировка и варианты	<code>(...)</code> , <code> </code>
экранирование	<code>\.</code> , <code>\+</code> , <code>\(</code>

Функции: `re.fullmatch` — проверка всей строки, `re.search` — поиск первого фрагмента (`.group()` , `.start()` , `.end()`), `re.sub` — замена всех фрагментов.

Возможные вопросы на экзамене:

1. Чем отличаются `re.fullmatch(r"\d+", s)` и `re.search(r"\d+", s)` ?
2. Почему шаблоны записывают в виде `r"..."` ?
3. Можно ли регулярным выражением проверить, что дата `29.02.2027` существует?

Дальше: `re.findall` , `re.split` , `re.finditer` (после темы «Списки»), извлечение групп, флаги, `re.compile` .

In []: