ГЛЕБ ЗАЙЦЕВ

Регулярные выражения

на Python

[A–Z]+

ПОИСК · ПРОВЕРКА · ПРЕОБРАЗОВАНИЕ

24 **задачи**
с проверкой
результата

Решения, контрпримеры
и границы применимости

СТАНДАРТНАЯ БИБЛИОТЕКА PYTHON

Глеб Зайцев

Регулярные выражения на Python. 24 задачи с проверкой результата

«Автор»
2026

Зайцев Г.

Регулярные выражения на Python. 24 задачи с проверкой результата / Г. Зайцев — «Автор», 2026

Практикум для тех, кто знает основы Python и хочет увереннее искать, проверять и преобразовывать текст. 24 задачи: коды документов, границы слов, Unicode, пробелы, группы, журналы событий, замены и разбиение строк. В каждой задаче есть точное условие, запускаемое решение, ожидаемый результат и дополнительные проверки, включая неподходящий ввод. Отдельно разобраны ограничения: почему совпадение с форматом даты не доказывает её существование и почему CSV лучше читать специальным парсером. Все примеры самостоятельны, работают в памяти и используют стандартную библиотеку Python. Аккаунты, платные сервисы и скачивание учебных данных не нужны.

Содержание

Начните с результата, а не с красивого шаблона 5
Как запускать и читать примеры 6
Три разных вопроса к строке 7
Задача 1. Проверить код документа целиком 8
Задача 2. Извлечь коды, не цепляя часть длинного идентификатора 9
Задача 3. Различить ASCII-цифры и Unicode-цифры 11
Задача 4. Искать пользовательскую строку буквально 12
Конец ознакомительного фрагмента. 13

Глеб Зайцев
Регулярные выражения на Python. 24 задачи с проверкой результата

Начните с результата, а не с красивого шаблона

Регулярное выражение полезно, когда заранее понятно, какой текст нужно найти и какой нельзя принять. Если условие звучит как «вытащи правильные номера», сначала придётся договориться, что считается номером: сколько в нём знаков, какие буквы разрешены и где заканчивается одно значение. Без этого даже безошибочный синтаксис может давать неправильный для задачи результат.

В книге 24 небольшие самостоятельные работы. Каждая начинается с ограничения формата и заканчивается проверками. Набор проверок включает не только удачный пример: пустую строку, лишний символ, похожую последовательность или другой крайний случай. Это помогает отличать решение задачи от совпадения, которое случайно сработало на одной строке.

Нужны базовые знания Python: строки, списки, словари и вызов функции. Используются короткие выражения включения списков. В них запись [действие for элемент in набор] означает собрать результат действия для каждого элемента. Мы не предполагаем, что вы уже знаете синтаксис regex. Он объясняется по мере появления в заданиях.

Все данные синтетические. Программы ничего не читают с диска, не отправляют в интернет и не изменяют рабочие документы. Запускайте их сначала как отдельные учебные файлы. Не вставляйте персональные данные или реальные журналы в незнакомый онлайн-тестер только ради удобства проверки.

Как запускать и читать примеры

Примеры проверяются на Python 3.12. Нужен обычный интерпретатор из официального источника python.org. Сторонних пакетов нет: re, csv и io входят в стандартную библиотеку. Создайте в новой учебной папке файл task.py, скопируйте полный блок решения, сохраните как UTF-8 и запустите python3 task.py. В системах, где интерпретатор вызывается иначе, используйте настроенную команду Python 3. В терминале должен появиться результат, приведённый в книге.

Код в каждом решении самостоятельный. Не требуется сначала выполнять прошлые задачи или загружать архив автора. Не включайте в файл заголовок главы и блок ожидаемого вывода. В исходнике каждая строка кода начинается с левого края: в этих примерах нет вложенных блоков с обязательными отступами. Если читалка добавила перед строками пробелы, в том числе неразрывные, удалите только эти начальные отступы. Пробелы внутри строк и сами переносы строк сохраняйте. Длинная строка может переноситься читалкой визуально; это не повод вставлять в выражение дополнительный Enter.

Перед чтением решения попробуйте сформулировать ответ сами. В разделе дополнительных проверок для каждого случая отдельно замените только указанное исходное значение в полном блоке программы и запустите его заново. Шаблон и порядок действий должны оставаться прежними. Эти проверки независимы друг от друга: не накапливайте изменения входных переменных между ними.

Переменная text хранит исходную строку. В некоторых заданиях используется samples — список вариантов. Pattern обозначает шаблон, result — вычисленный ответ. Print(repr(result)) выводит представление ответа: символы новой строки становятся видимыми как \n, табуляции — как \t. Это позволяет сравнить не только слова, но и невидимые разделители. Кавычки вокруг напечатанной строки не обязательно являются её содержимым.

Большинство шаблонов записаны как raw-строки с буквой r перед кавычкой. Например, r'\b' передаёт модулю re обратную косую черту и букву b, а не служебный символ обычной Python-строки. Raw-строка не отменяет правила regex: она только помогает передать нужный текст шаблона через синтаксис Python.

Книга подготовлена с помощью ИИ. Полные блоки решений выполнены, а результаты сопоставлены с заранее заданными ответами и дополнительными случаями. Проверки не являются математическим доказательством правильности для любого ввода. У каждого шаблона остаются явные ограничения; версия интерпретатора и результаты локальной проверки фиксируются при сборке рукописи.

Три разных вопроса к строке

Fullmatch отвечает на вопрос, подходит ли вся строка целиком. Search ищет хотя бы одно совпадение где-то внутри. Findall возвращает найденные фрагменты, но форма результата зависит от захватывающих групп. Подмена одного вопроса другим — частый источник ошибок даже при хорошем шаблоне.

Для проверки одного идентификатора обычно нужен fullmatch. Для поиска кодов в письме — извлечение нескольких совпадений. Для изменения пробелов — sub. Выбор операции делается по условию задачи, а не по тому, какая функция первой встретилась в примере из интернета.

Ни одна из этих операций не доказывает истинность сведений. Формально корректный код может отсутствовать в базе, строка похожей даты — обозначать невозможный день, а имя — принадлежать кому угодно. Regex отвечает за описанный текстовый формат, а не за бизнес-смысл и права доступа.

Задача 1. Проверить код документа целиком

Формат учебного кода: две заглавные латинские буквы, дефис и ровно четыре ASCII-цифры. Пробелы, строчные буквы и посторонние символы запрещены. Верните True или False для каждого элемента samples; не вырезайте подходящую часть из неподходящего значения.

Решение

```
import re
samples = ['AB-0042', 'ab-0042', 'AB-0042x', '', 'AB-123']
pattern = r'[A-Z]{2}-[0-9]{4}'
result = [re.fullmatch(pattern, s) is not None for s in samples]
print(repr(result))
```

Ожидаемый результат

```
[True, False, False, False, False]
```

Квадратные скобки задают допустимые символы, фигурные — число повторов. [A-Z]{2} означает два символа из указанного латинского диапазона, а не две произвольные буквы любого языка. Дефис вне класса здесь обозначает обычный дефис.

Мы используем fullmatch, потому что проверяем значение целиком. Search мог бы найти AB-0042 внутри более длинной строки и тем самым ответить не на заданный вопрос. Нормализация пробелов не выполняется: по условию они запрещены, поэтому молча исправлять вход нельзя.

Дополнительные проверки

```
samples = ['ZZ-9999', ' AA-0001', 'AA-0001\n']
```

Должно получиться:

```
[True, False, False]
```

Вопрос

Нужно ли добавить ^ и $ к этому шаблону для fullmatch?

Ответ

Нет, сама операция требует совпадения со всей строкой. Явные дополнительные якоря здесь не дают нужного нового ограничения.

Задача 2. Извлечь коды, не цепляя часть длинного идентификатора

В сообщении ищутся коды INV- и четыре ASCII-цифры. Соседние буквы, цифры и подчёркивания любого языка означают, что это часть другого идентификатора. Разделителями могут быть пробелы или пунктуация. Верните все самостоятельные коды в исходном порядке.

Решение

```
    import re
    text   =   'INV-0042;   XINV-0042;   INV-00421;   (INV-0700);
INV-0001_extra'
    pattern = r'(?<!\w)INV-[0-9]{4}(?!\w)'
    result = re.findall(pattern, text)
    print(repr(result))
```

Ожидаемый результат

```
    ['INV-0042', 'INV-0700']
```

Отрицательная проверка позади (?<!...) запрещает непосредственно предыдущий символ заданного класса, а (?!...) — следующий. Эти проверки не забирают разделитель в результат. Поэтому соседние коды через slash находятся независимо.

В стандартном режиме Python класс \w включает Unicode-буквы, цифры и подчёркивание. Это соответствует сформулированному здесь условию, но не любой задаче поиска. Если соседство русского слова должно быть разрешено, условие и проверка границ потребуют другого решения.

Дополнительные проверки

```
    text = 'яINV-1234 INV-5678я'
```

Должно получиться:

```
    []
    text = 'INV-0000/INV-0001'
```

Должно получиться:

```
    ['INV-0000', 'INV-0001']
    text = ''
```

Должно получиться:

```
    []
```

Вопрос

Почему шаблон без правой проверки нашёл бы INV-0042 внутри INV-00421?

Ответ

Требование четырёх цифр описывает совпадение, но само по себе не запрещает пятую цифру после него. Граница — отдельное условие.

Задача 3. Различить ASCII-цифры и Unicode-цифры

Сравните два контракта: строка состоит из трёх десятичных цифр Unicode или только из трёх знаков 0–9. В примере есть арабско-индийские и полноширинные цифры. Возвратите два списка логических результатов, не преобразуя строки в числа.

Решение

```
import re
samples = ['123', '١٢٣', '###', '12A']
result = ([re.fullmatch(r'\d{3}', s) is not None for s in samples], [re.fullmatch(r'[0-9]{3}', s) is not None for s in samples])
print(repr(result))
```

Ожидаемый результат

```
([True, True, True, False], [True, False, False, False])
```

Короткая запись \d не всегда означает ровно набор ASCII 0–9. В Unicode-строках Python она охватывает десятичные цифры Unicode. При этом далеко не каждый символ, похожий на число, относится к этому классу: надстрочные цифры в дополнительной проверке не принимаются.

Выбор зависит от данных. Для международного текста широкий класс может быть полезен. Для внешнего формата, который требует конкретные байты ASCII, лучше выразить требование явно. Ни один вариант не следует объявлять универсально правильным без описания входа.

Дополнительные проверки

```
samples = ['000', '²³⁴', '']
```

Должно получиться:

```
([True, False, False], [True, False, False])
```

Вопрос

Можно ли по внешнему виду цифры на экране определить, что хранится знак ASCII?

Ответ

Надёжно — нет. Похожие символы могут иметь разные коды Unicode. Формат нужно проверять по реальному значению строки.

Задача 4. Искать пользовательскую строку буквально

Найдите буквальное имя a+b.txt внутри текста. Плюс и точка должны означать именно эти символы, а не операторы regex. В этой работе мы ищем подстроку, а не проверяем границы имени файла и не открываем файл.

Решение

```
import re
text = 'a+b.txt ab.txt aaabXtxt a+b.txt'
needle = 'a+b.txt'
```

Конец ознакомительного фрагмента.