



Больше информации по Python тут



Наталья Кайда 24 октября 2022



4



3



0



0



Самоучитель по Python для начинающих. Часть 4. Методы работы со строками

Разберем самые полезные методы работы с текстовыми данными: узнаем, как искать, заменять и подсчитывать символы, конвертировать регистр и определять, из каких элементов состоят строки. В конце статьи – 10 задач для тренировки.



Обсудить



3



4

4

← [Часть 3](#)

Текстовые переменные str в Питоне

Строковый тип `str` в Python используют для работы с любыми текстовыми данными. Python автоматически определяет тип `str` по кавычкам – одинарным или двойным:

```
>>> stroka = 'Python'
>>> type(stroka)
<class 'str'>
>>> stroka2 = "code"
>>> type(stroka2)
<class 'str'>
```

Для решения многих задач строковую переменную нужно объявить заранее, до начала исполнения основной части программы. Создать пустую переменную `str` просто:

```
stroka = ''
```

Или:

```
stroka2 = ""
```

Если в самой строке нужно использовать кавычки – например, для названия книги – то один вид кавычек используют для строки, второй – для выделения названия:

```
>>> print("'Самоучитель Python' - возможно, лучший справочник по Питону.")
'Самоучитель Python' - возможно, лучший справочник по Питону.
>>> print('"Самоучитель Python" - возможно, лучший справочник по Питону.')
"Самоучитель Python" - возможно, лучший справочник по Питону.
```

Использование одного и того же вида кавычек внутри и снаружи строки вызовет ошибку:

```
>>> print('"Самоучитель Python" - возможно, лучший справочник по Питону.')
  File "<ipython-input-1-1.py"> line 1
    '"Самоучитель Python" - возможно, лучший справочник по Питону.'
```

SyntaxError: invalid syntax

Кроме двойных " и одинарных кавычек ' , в Python используются и тройные ''' – в них заключают текст, состоящий из нескольких строк, или программный код:

```
>>> print('''В тройные кавычки заключают многострочный текст.  
Программный код также можно выделить тройными кавычками.''' )  
В тройные кавычки заключают многострочный текст.  
Программный код также можно выделить тройными кавычками.
```

Длина строки len в Python

Для определения длины строки используется встроенная функция **len()**. Она подсчитывает общее количество символов в строке, включая пробелы:

```
>>> stroka = 'python'  
>>> print(len(stroka))  
6  
>>> stroka1 = ' '  
>>> print(len(stroka1))  
1
```

Преобразование других типов данных в строку

Целые и вещественные числа преобразуются в строки одинаково:

```
>>> number1 = 55  
>>> number2 = 55.5  
>>> stroka1 = str(number1)  
>>> stroka2 = str(number2)  
>>> print(type(stroka1))  
<class 'str'>  
>>> print(type(stroka2))  
<class 'str'>
```

Решение многих задач значительно упрощается, если работать с числами в строковом формате.

Сложение и умножение строк

Как уже упоминалось в [предыдущей главе](#), строки можно складывать – эта операция также известна как **конкатенация**:

```
>>> str1 = 'Python'
>>> str2 = ' - '
>>> str3 = 'самый гибкий язык программирования'
>>> print(str1 + str2 + str3)
Python - самый гибкий язык программирования
```

При необходимости строку можно умножить на целое число – эта операция называется **репликацией**:

```
>>> stroka = '*** '
>>> print(stroka * 5)
*** *** *** *** ***
```

Подстроки

Подстрокой называется фрагмент определенной строки. Например, **'abra'** является подстрокой **'abrakadabra'**. Чтобы определить, входит ли какая-то определенная подстрока в строку, используют оператор `in`:

```
>>> stroka = 'abrakadabra'
>>> print('abra' in stroka)
True
>>> print('zebra' in stroka)
False
```

Индексация строк в Python

Для обращения к определенному символу строки используют индекс – порядковый номер элемента. Python поддерживает два типа индексации – **положительную**, при которой отсчет элементов начинается с **0** и **с начала** строки, и **отрицательную**, при которой отсчет начинается с **-1** и **с конца**:

| | | | | | | | |
|-----------------------|---|---|---|---|---|---|---|
| Положительные индексы | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|-----------------------|---|---|---|---|---|---|---|

Наш сайт использует файлы cookie для вашего максимального удобства.
Пользуясь сайтом, вы даете свое согласие с [условиями пользования cookie](#)

Согласен

Чтобы получить определенный элемент строки, нужно указать его индекс в квадратных скобках:

```
>>> stroka = 'программирование'
>>> print(stroka[7])
м
>>> print(stroka[-1])
е
```

Срезы строк в Python

Индексы позволяют работать с **отдельными** элементами строк. Для работы с **подстроками** используют **срезы**, в которых задается нужный диапазон:

```
>>> stroka = 'программирование'
>>> print(stroka[7:10])
мир
```

Диапазон среза `[a:b]` начинается с первого указанного элемента **a** **включительно**, и заканчивается на последнем, **не включая b** в результат:

```
>>> stroka = 'программа'
>>> print(stroka[3:8])
грамм
```

Если не указать первый элемент диапазона `[:b]`, срез будет выполнен с начала строки до позиции второго элемента **b**:

```
>>> stroka = 'программа'
>>> print(stroka[:4])
прог
```

В случае отсутствия второго элемента `[a:]` срез будет сделан с позиции первого символа и до конца строки:

Если не указана ни стартовая, ни финальная позиция среза, он будет равен **исходной** строке:

```
>>> stroka = 'позиции не заданы'
>>> print(stroka[:])
позиции не заданы
```

Шаг среза

Помимо диапазона, можно задавать **шаг** среза. В приведенном ниже примере выбирается символ из стартовой позиции среза, а затем каждая 3-я буква из диапазона:

```
>>> stroka = 'Python лучше всего подходит для новичков.'
>>> print(stroka[1:15:3])
уолшв
```

Шаг может быть отрицательным – в этом случае символы будут выбираться, начиная с конца строки:

```
>>> stroka = 'это пример отрицательного шага'
>>> print(stroka[-1:-15:-4])
а нт
```

Срез `[::-1]` может оказаться очень полезным при решении задач, связанных с палиндромами:

```
>>> stroka = 'А роза упала на лапу Азора'
>>> print(stroka[::-1])
арозА упал ан алапу азор А
```

Замена символа в строке

Строки в Python относятся к **неизменяемым** типам данных. По этой причине попытка замены символа по индексу обречена на провал:

```
>>> stroka = 'wall'
```

```
File "<pyshell>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Но заменить любой символ все-таки можно – для этого придется воспользоваться срезами и конкатенацией. Результатом станет новая строка:

```
>>> stroka = 'mall'
>>> stroka = 'b' + stroka[1:]
>>> print(stroka)
ball
```

Более простой способ «замены» символа или подстроки – использование метода **replace()**, который мы рассмотрим ниже.

Полезные методы строк

Python предоставляет множество методов для работы с текстовыми данными. Все методы можно сгруппировать в четыре категории:

- Преобразование строк.
- Оценка и классификация строк.
- Конвертация регистра.
- Поиск, подсчет и замена символов.

Рассмотрим эти методы подробнее.

Преобразование строк

Три самых используемых метода из этой группы – **join()**, **split()** и **partition()**. Метод **join()** незаменим, если нужно преобразовать список или кортеж в строку:

```
>>> spisok = ['Я', 'изучаю', 'Python']
>>> stroka = ' '.join(spisok)
>>> print(stroka)
Я изучаю Python
```

При объединении списка или кортежа в строку можно использовать любые разделители:

```
>>> print(stroka)
Я***изучаю***Django
```

Метод **split()** используется для обратной манипуляции – преобразования строки в список:

```
>>> text = 'это пример текста для преобразования в список'
>>> spisok = text.split()
>>> print(spisok)
['это', 'пример', 'текста', 'для', 'преобразования', 'в', 'список']
```

По умолчанию **split()** разбивает строку по пробелам. Но можно указать любой другой символ – и на практике это часто требуется:

```
>>> text = 'цвет: синий; вес: 1 кг; размер: 30x30x50; материал: картон'
>>> spisok = text.split(';')
>>> print(spisok)
['цвет: синий', ' вес: 1 кг', ' размер: 30x30x50', ' материал: картон']
```

Метод **partition()** поможет преобразовать строку в кортеж:

```
>>> text = 'Python - простой и понятный язык'
>>> kort = text.partition('и')
>>> print(kort)
('Python - простой ', 'и', ' понятный язык')
```

В отличие от **split()**, **partition()** учитывает только **первое** вхождение элемента-разделителя (и добавляет его в итоговый кортеж).

Оценка и классификация строк

В Python много встроенных методов для оценки и классификации текстовых данных. Некоторые из этих методов работают только со строками, в то время как другие универсальны. К последним относятся, например, функции **min()** и **max()**:

```
>>> text = '12345'
>>> print(min(text))
1
```


В Python есть специальные методы для определения **типа** символов. Например, **isalnum()** оценивает, состоит ли строка из **букв и цифр**, либо в ней есть какие-то другие символы:

```
>>> text = 'abracadabra123456'
>>> print(text.isalnum())
True
>>> text1 = 'a*b$ra cadabra'
>>> print(text1.isalnum())
False
```

Метод **isalpha()** поможет определить, состоит ли строка **только из букв**, или включает специальные символы, пробелы и цифры:

```
>>> text = 'программирование'
>>> print(text.isalpha())
True
>>> text2 = 'password123'
>>> print(text2.isalpha())
False
```

С помощью метода **isdigit()** можно определить, входят ли в строку **только цифры**, или там есть и другие символы:

```
>>> text = '1234567890'
>>> print(text.isdigit())
True
>>> text2 = '123456789o'
>>> print(text2.isdigit())
False
```

Поскольку вещественные числа содержат точку, а отрицательные – знак минуса, выявить их этим методом не получится:

```
>>> text1 = '-5'
>>> print(text1.isdigit())
False
```

Если нужно определить наличие в строке дробей или римских цифр, подойдет метод **isnumeric()**:

```
>>> text = '½¼⅓⅔⅕'
>>> print(text.isdigit())
False
>>> print(text.isnumeric())
True
```

Методы **islower()** и **isupper()** определяют регистр, в котором находятся буквы. Эти методы игнорируют небуквенные символы:

```
>>> text = 'abracadabra'
>>> print(text.islower())
True
>>> text2 = 'Python bytes'
>>> print(text2.islower())
False
>>> text3 = 'PYTHON'
>>> print(text3.isupper())
True
```

Метод **isspace()** определяет, состоит ли анализируемая строка из одних пробелов, или содержит что-нибудь еще:

```
>>> stroka = '   '
>>> print(stroka.isspace())
True
>>> stroka2 = '  a  '
>>> print(stroka2.isspace())
False
```

Конвертация регистра

Из всех методов, связанных с конвертацией регистра, наиболее часто используются на практике два – **lower()** и **upper()**. Они преобразуют все символы в **нижний** и **верхний** регистр соответственно:

```
>>> text = 'этот текст надо написать заглавными буквами'
>>> print(text.upper())
ЭТОТ ТЕКСТ НАДО НАПИСАТЬ ЗАГЛАВНЫМИ БУКВАМИ
```



здесь все буквы разные, а нужны прописные

Иногда требуется преобразовать текст так, чтобы с заглавной буквы начиналось только первое слово предложения:

```
>>> text = 'предложение должно начинаться с ЗАГЛАВНОЙ буквы.'
>>> print(text.capitalize())
```

Предложение должно начинаться с заглавной буквы.

Методы **swapcase()** и **title()** используются реже. Первый заменяет исходный регистр на противоположный, а второй – начинает каждое слово с заглавной буквы:

```
>>> text = 'ПРИМЕР ИСПОЛЬЗОВАНИЯ swapcase'
>>> print(text.swapcase())
Пример Ипользования SWAPCASE
>>> text2 = 'тот случай, когда нужен метод title'
>>> print(text2.title())
Тот Случай, Когда Нужен Метод Title
```

Поиск, подсчет и замена символов

Методы **find()** и **rfind()** возвращают индекс стартовой позиции искомой подстроки. Оба метода учитывают только **первое** вхождение подстроки. Разница между ними заключается в том, что **find()** ищет первое вхождение подстроки **с начала** текста, а **rfind()** – **с конца**:

```
>>> text = 'пример текста, в котором нужно найти текстовую подстроку'
>>> print(text.find('текст'))
7
```

Такие же результаты можно получить при использовании методов **index()** и **rindex()** – правда, придется предусмотреть обработку ошибок, если искомая подстрока не будет обнаружена:

```
>>> text = 'Съешь еще этих мягких французских булок!'
>>> print(text.index('еще'))
6
>>> print(text.rindex('чаю'))
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
ValueError: substring not found
```

Если нужно определить, начинается ли строка с определенной подстроки, поможет метод **startswith()**:

```
>>> text = 'Жила-была курочка Ряба'
>>> print(text.startswith('Жила'))
True
```

Чтобы проверить, заканчивается ли строка на нужное окончание, используют **endswith()**:

```
>>> text = 'В конце всех ждал хэппи-энд'
>>> print(text.endswith('энд'))
True
```

Для подсчета числа вхождений определенного символа или подстроки применяют метод **count()** – он помогает подсчитать как общее число вхождений в тексте, так и вхождения в указанном диапазоне:

```
>>> text = 'Съешь еще этих мягких французских булок, да выпей же чаю!'
>>> print(text.count('е'))
5
>>> print(text.count('е', 5, 25))
2
```

Методы **strip()**, **lstrip()** и **rstrip()** предназначены для удаления пробелов. Метод **strip()** удаляет пробелы в начале и конце строки, **lstrip()** – только **слева**, **rstrip()** – только **справа**:

```
>>> text = '    здесь есть пробелы и слева, и справа    '
>>> print('***', text.strip(), '***')
*** здесь есть пробелы и слева, и справа ***
>>> print('***', text.lstrip(), '***')
*** здесь есть пробелы и слева, и справа ***
>>> print('***', text.rstrip(), '***')
***    здесь есть пробелы и слева, и справа ***
```

Метод **replace()** используют для замены символов или подстрок. Можно указать нужное **количество замен**, а сам символ можно заменить на пустую подстроку – проще говоря, **удалить**:

```
>>> text = 'В этой строчке нужно заменить только одну "ч"'
>>> print(text.replace('ч', '', 1))
В этой строке нужно заменить только одну "ч"
```

Стоит заметить, что метод **replace()** подходит лишь для самых простых вариантов замены и удаления подстрок. В более сложных случаях необходимо использование регулярных выражений, которые мы будем изучать позже.

Практика

Задание 1

Напишите программу, которая получает на вход строку и выводит:

- количество символов, содержащихся в тексте;
- **True** или **False** в зависимости от того, являются ли все символы буквами и цифрами.

Решение:

```
text = input()
print(len(text))
print(text.isalpha())
```

Задание 2

Напишите программу, которая получает на вход слово и выводит **True**, если слово является палиндромом, или **False** в противном случае. Примечание: для сравнения в Python используется

```
text = input().lower()
print(text == text[::-1])
```

Задание 3

Напишите программу, которая получает строку с именем, отчеством и фамилией, написанными в произвольном регистре, и выводит данные в правильном формате. Например, строка **алеКСандр СЕРГЕЕВИЧ ПушкиН** должна быть преобразована в **Александр Сергеевич Пушкин**.

Решение:

```
text = input()
print(text.title())
```

Задание 4

Имеется строка **12361573928167047230472012**. Напишите программу, которая преобразует строку в текст **один236один573928один670472304720один2**.

Решение:

```
text = '12361573928167047230472012'
print(text.replace('1', 'один'))
```

Задание 5

Напишите программу, которая последовательно получает на вход имя, отчество, фамилию и должность сотрудника, а затем преобразует имя и отчество в инициалы, добавляя должность после запятой.

Пример ввода:

```
Алексей
Константинович
Романов
бухгалтер
```

Вывод:

Наш сайт использует файлы cookie для вашего максимального удобства.
Пользуясь сайтом, вы даете свое согласие с [условиями пользования cookie](#)

Согласен

Решение:

```
first_name, patronymic, last_name, position = input(), input(), input(), input()
print(first_name[0] + '.', patronymic[0] + '.', last_name + ',', position)
```

Задание 6

Напишите программу, которая получает на вход строку текста и букву, а затем определяет, встречается ли данная буква (в любом регистре) в тексте. В качестве ответа программа должна выводить **True** или **False**.

Пример ввода:

```
ЗонТИК
к
```

Вывод:

```
True
```

Решение:

```
text = input().lower()
letter = input()
print(letter in text)
```

Задание 7

Напишите программу, которая определяет, является ли введенная пользователем буква **гласной**. В качестве ответа программы выводит **True** или **False**, буквы могут быть как в верхнем, так и в нижнем регистре.

Решение:

```
vowels = 'аиеёоуыэюя'
text = input().lower()
```

Задание 8

Напишите программу, которая принимает на вход строку текста и подстроку, а затем выводит индексы первого вхождения подстроки с начала и с конца строки (без учета регистра).

Пример ввода:

Шесть шустрых мышат в камышах шуршат
ша

Вывод:

16 33

Решение:

```
text, letter = input().lower(), input()
print(text.find(letter), text.rfind(letter))
```

Задание 9

Напишите программу для подсчета количества пробелов и непробельных символов в введенной пользователем строке.

Пример ввода:

В роще, травы шевеля, мы нащиплем щавеля

Вывод:

Количество пробелов: 6, количество других символов: 34

Решение:

```
text = input()
nospace = text.replace(' ', '')
```

Наш сайт использует файлы cookie для вашего максимального удобства.
Пользуясь сайтом, вы даете свое согласие с [условиями пользования cookie](#)

Согласен

Задание 10

Напишите программу, которая принимает строку и две подстроки **start** и **end**, а затем определяет, начинается ли строка с фрагмента **start**, и заканчивается ли подстрокой **end**. Регистр не учитывать.

Пример ввода:

```
Программирование на Python - лучшее хобби
про
про
```

Вывод:

```
True
False
```

Решение:

```
text, start, end = input().lower(), input(), input()
print(text.startswith(start))
print(text.endswith(end))
```

Подведем итоги

В этой части мы рассмотрели самые популярные методы работы со строками – они пригодятся для решения тренировочных задач и в разработке реальных проектов. В следующей статье будем разбирать методы работы со списками.



Содержание самоучителя

1. Особенности, сферы применения, установка, онлайн IDE
2. Все, что нужно для изучения Python с нуля – книги, сайты, каналы и курсы
3. Типы данных: преобразование и базовые операции
4. Методы работы со строками

7. Методы работы с кортежами
8. Методы работы со множествами
9. Особенности цикла for
10. Условный цикл while
11. Функции с позиционными и именованными аргументами

Материалы по теме

- [ТОП-15 трюков в Python 3, делающих код понятнее и быстрее](#)

♥ 4 💬 Обсудить 📌 3 🔥 3 💧 0 💩 0

Python



МЕРОПРИЯТИЯ

Delivery Meetup SPB #2

📅 26 января [Онлайн](#) [Бесплатно](#)

Цифровое импортозамещение

📅 24 января [Онлайн](#) [Бесплатно](#)

Миграция на отечественный почтовый сервер TEGU

📅 02 февраля [Онлайн](#) [Бесплатно](#)

+ Показать еще

Комментарии

Оставьте свой комментарий (можно использовать markdown)



Отправить

Наш сайт использует файлы cookie для вашего максимального удобства.
Пользуясь сайтом, вы даете свое согласие с [условиями пользования cookie](#)

Согласен

Data scientist

Москва, от 172000 RUB до 230000 RUB

UI/UX дизайнер

Москва, от 120000 RUB до 150000 RUB

C# developer

Санкт-Петербург, от 2000 USD до 4000 USD

+ Показать еще

Опубликовать вакансию

ЛУЧШИЕ СТАТЬИ ПО ТЕМЕ

ООП на Python: концепции, принципы и примеры реализации

Программирование на Python допускает различные методологии, но в его основе лежит объектный подход, поэтому работать в стиле ООП на Python очень просто.

3 самых важных сферы применения Python: возможности языка

Существует множество областей применения Python, но в некоторых он особенно хорош. Разбираемся, что же можно делать на этом ЯП.

Программирование на Python: от новичка до профессионала

Пошаговая инструкция для всех, кто хочет изучить программирование на Python (или программирование вообще), но не знает, куда сделать первый шаг.

О проекте

Реклама

Пользовательское соглашение

Наш сайт использует файлы cookie для вашего максимального удобства.
Пользуясь сайтом, вы даете свое согласие с [условиями пользования cookie](#)

Согласен

Контакты

☐ Push-уведомления

☐ Темная тема



© 2023, Proglib. При копировании материала ссылка на источник обязательна.