**проРабота со строками**

Со строками в Python можно выполнять различные операции. Также существует модуль string, который можно импортировать для получения доступа к еще большим возможностям. Пройдемся по следующим разделам:

* Как создавать строки
* Конкатенация строк
* Умножение строк
* Методы строк
* Замена строк

**Создание строки**

Строки всегда создаются одним из трех способов. Вы можете использовать одинарные, двойные (кавычки) и тройные апострофы:

my\_string = "Добро пожаловать в Python!"

print(my\_string)

another\_string = 'Я новый текст тут...'

print (another\_string)

a\_long\_string = '''А это у нас новая строка в троичных скобках'''

print (a\_long\_string)

Данный код демонстрирует то, как вы можете использовать одинарные или двойные скобки в строку.

my\_string = "I'm a Python programmer!"

print(my\_string)

otherString = 'Слово "Python" обычно подразумевает змею'

print(otherString)

tripleString = """В такой "строке" мы можем 'использовать' все."""

print(tripleString)

Существует еще один способ создания строки, при помощи **метода str:**

my\_number = 123

print(my\_number)

my\_string = str(my\_number)

print(my\_string)

Cтрока – это один из неизменных **типов Python**. Это значит, что вы не можете менять содержимое строки после ее создания. В Python, начиная с версии 3, все **строки являются юникодом**.

**Конкатенация строк**

Конкатенация – это соединение или добавление двух объектов вместе. Оператор **+** конкатенирует две строки в одну

*my\_string = 'Python -' + ' лучший язык программирования!'  
print(my\_string)  
Python - лучший язык программирования!  
  
string1 = 'Сервер упал'  
string2 = 'Звоните Васе!'  
string1 = 'Сервер упал.'  
string2 = 'Звоните Васе!'  
alert\_message = string1 + string2  
print(alert\_message)  
Сервер упал.Звоните Васе!*

**Оператор умножения строк \***

*my\_string = 'go '  
my\_string = my\_string\*10  
  
print(my\_string)  
go go go go go go go go go go*

**Методы строк**

Строка является **объектом** в Python. Строки содержат собственные встроенные методы.

Например:

|  |  |
| --- | --- |
| 1 | my\_string = "This is a string!" |

Теперь вам нужно сделать так, чтобы вся эта строка была в верхнем регистре. Для этого нужно вызвать метод **upper**():

my\_string = "This is a string!"

print(my\_string)

print(my\_string.upper())

Если вам нужно, чтобы все было в **нижнем регистре**, используйте метод **lower**(). Если вы хотите удалить все начальные и конечные пробелы, используйте метод **strip**().

Получение списка всех методов строк:

my\_string = "This is a string!"

print(dir(my\_string))

Вы можете игнорировать методы, которые начинаются и заканчиваются двойным подчеркиванием, например **\_\_add\_\_**. Они не используются в ежедневном программировании в Python. Лучше обратите внимание на другие. Если вы хотите узнать, что делает тот или иной метод, просто обратитесь к справке.

Например, если вы хотите узнать, зачем вам **capitalize**, впишите следующее, чтобы узнать:

my\_string = "This is a string!"

help(my\_string.capitalize)

**Нарезка строк**

Часто используется **нарезка строк**.

|  |  |
| --- | --- |
| 1 | my\_string = "I like Python!" |

Каждый символ в строке может стать доступным при помощи слайсинга (**нарезки**). Взять только первый символ, вы можете сделать это следующим образом:

my\_string = "This is a string!"

print( my\_string[0:1] ) # T

Здесь мы берем первый символ в строке до второго символа, но, не включая его. В Python **отсчет ведется с нуля**.

my\_string = "This is a string!"

print( my\_string[0:10] ) # This is a

Мы можем назначить срез, лишь **указав его начало** (т.е., *my\_string[2:]*), конец среза (*my\_string[:1]*), или оба (*my\_string[0:13]*). Мы можем использовать **отрицательные значения**, которые начинаются с конца строки. Так что в примере, где мы указали *my\_string[0:-5]*, начало ведется с нуля и заканчивается 5 символами, перед концом строки.

my\_string = "This is a string!"

print( my\_string[0] ) # T

Данный код выдаст первый символ в строке.

**Форматирование строк**

Форматирование строк – это замещение значений в базовой строке. Большую часть времени вы будете вставлять строки внутри строк, однако, вам также понадобиться вставлять целые числа и числа с запятыми в строки весьма часто. Существует два способа достичь этой цели. Начнем со старого способа, после чего перейдем к новому:

# -\*- coding: utf-8 -\*-

my\_string = "Я люблю %s" % "Python"

print(my\_string) # Я люблю Python

var = "Perl"

newString = "Я люблю %s" % var

print(newString) # Я люблю Perl

another\_string = "Я люблю %s и %s" % ("Python", var)

print(another\_string) # Я люблю Python и Perl

Cимвол **%** указывает Python, что вы скоро вставите текст на его место. Если вы будете следовать за строкой со знаком процента и другой строкой или переменной, Python попытается вставить ее в строку. Вы можете вставить несколько строк, добавив несколько знаков процента в свою строку. Когда вы добавляете больше одной строки, вам нужно **закрыть эти строки в круглые скобки**.

my\_string = "%i + %i = %i" % (1,2,3)

print(my\_string) # '1 + 2 = 3'

float\_string = "%f" % (1.23)

print(float\_string) # '1.230000'

float\_string2 = "%.2f" % (1.23)

print(float\_string2) # '1.23'

float\_string3 = "%.2f" % (1.237)

print(float\_string3) # '1.24'

Мы создали строку, которая принимает три аргумента, и мы передаем их.

Во втором примере, мы передаем число с запятой. Обратите внимание на то, что результат включает множество дополнительных нулей (1.230000). Нам это не нужно, так что мы указываем Python ограничить выдачу до двух десятичных значений в третьем примере (“**%.2f**”). Последний пример показывает, что Python **округлит числа** для вас, если вы передадите ему дробь, что лучше, чем два десятичных значения.

**Шаблоны и новая методика форматирования строк**

print("%(lang)s is fun!" % {"lang":"Python"}) # Python is fun!

Вторая часть примера вызывает словарь Python. Пара **key:value**, означает, что когда Python ищет ключ **lang** в строке и в указанном словаре ключей, он заменяет этот ключ его значением. Давайте взглянем на следующие примеры:

a = "Python is as simple as {0}, {1}, {2}".format("a", "b", "c")

print(a) # 'Python is as simple as a, b, c'

b = "Python is as simple as {1}, {0}, {2}".format("a", "b", "c")

print(b) # 'Python is as simple as b, a, c'

xy = {"x":0, "y":10}

c = "Graph a point at where x={x} and y={y}".format(\*\*xy)

print(c) # Graph a point at where x=0 and y=10

Дополнительно

* [Официальная документация Python о типе str](https://docs.python.org/3/library/functions.html#func-str)
* [Форматирование строк](https://docs.python.org/3/library/string.html#string-formatting)
* [Больше о форматировании строк](https://docs.python.org/3/library/string.html#formatexamples)
* Документация Python 2.Х о [юникоде](http://docs.python.org/2/library/functions.html#unicode)

# Конкатенация (сложение)

s1 = 'spam'

s2 = 'eggs'

print(s1 + s2)

# 'spameggs'

# Дублирование строки

print('spam' \* 3)

# spamspamspam

# Длина строки

print(len('spam'))

# 4

#### **Методы find и rfind**

Метод find находит в данной строке (к которой применяется метод) данную подстроку (которая передается в качестве параметра). Функция возвращает индекс первого вхождения искомой подстроки. Если же подстрока не найдена, то метод возвращает значение -1.

S = 'Hello'

print(S.find('e'))

# вернёт 1

print(S.find('ll'))

# вернёт 2

print(S.find('L'))

# вернёт -1

Метод rfind возвращает индекс последнего вхождения данной строки (“поиск справа”):

S = 'Hello'

print(S.find('l'))

# вернёт 2

print(S.rfind('l'))

# вернёт 3

Если вызвать метод find с тремя параметрами S.find(T, a, b), то поиск будет осуществляться в срезе S[a:b]. Если указать только два параметра S.find(T, a), то поиск будет осуществляться в срезе S[a:], то есть начиная с символа с индексом a и до конца строки. Метод S.find(T, a, b) возращает индекс в строке S, а не индекс относительно среза.

#### **Метод replace**

Метод replace заменяет все вхождения одной строки на другую. Формат:S.replace(old, new) — заменить в строке S все вхождения подстроки old на подстроку new. Пример:

print('Hello'.replace('l', 'L'))

# вернёт 'HeLLo'

Если вызвать метод find с тремя параметрами S.find(T, a, b), то поиск будет осуществляться в срезе S[a:b]. Если указать только два параметра S.find(T, a), то поиск будет осуществляться в срезе S[a:], то есть начиная с символа с индексом a и до конца строки. Метод S.find(T, a, b) возращает индекс в строке S, а не индекс относительно среза.

Если методу replace задать еще один параметр: S.replace(old, new, count), то заменены будут не все вхождения, а только не больше, чем первые count из них.

print('Abrakadabra'.replace('a', 'A', 2))

# вернёт 'AbrAkAdabra'

#### **Метод count**

Подсчитывает количество вхождений одной строки в другую строку. Простейшая форма вызова S.count(T) возвращает число вхождений строки T внутри строки S. При этом подсчитываются только непересекающиеся вхождения, например:

print('Abracadabra'.count('a'))

# вернёт 4

print(('a' \* 10).count('aa'))

# вернёт 5

При указании трех параметров S.count(T, a, b), будет выполнен подсчет числа вхождений строки T в срезе S[a:b].

### **Использование для строки функции «join»**

Функция join является более гибким способом объединения строк. С помощью функции join вы можете добавить в строку любой символ.

Например, если вы хотите добавить двоеточие (:) после каждого символа в строке «Python», вы можете использовать следующий код.

print(":".join("Python"))

Пример

var1 = "Guru99!"

var2 = "Software Testing"

print ("var1[0]:",var1[0])

print ("var2[1:5]:",var2[1:5])

#Some more examples

x = "Hello World!"

print (x[:6])

print (x[0:6] + "Guru99")

#Python String replace() Method

oldstring = 'I like Guru99'

newstring = oldstring.replace('like', 'love')

print (newstring)

#Changing upper and lower case strings

string="python at guru99"

print (string.upper())

string="python at guru99"

print (string.capitalize())

string="PYTHON AT GURU99"

print (string.lower())

#Using "join" function for the string

print(":".join("Python"))

#Reversing String

string="12345"

print(''.join(reversed(string)))

#Split Strings

word="guru99 career guru99"

print (word.split(' '))

word="guru99 career guru99"

print (word.split('r'))

x = "Guru99"

x.replace("Guru99","Python")

print (x)

##### **Таблица типов форматирования для строк**

| **Код** | **Значение** |
| --- | --- |
| **s** | Строковый |
| **r** | Строковый, но с использованием repr, а не str |
| **c** | Посимвольный |
| **d** | Десятичный |
| **i** | Целый |
| **u** | То же, что и d (no longer unsigned) |
| **o** | Восьмеричный |
| **x** | Шестнадцатеричный |
| **X** | Шестнадцатеричный в верхнем регистре |
| **e** | Floating-point exponent, нижний регистр |
| **E** | То же, что и e, но в верхнем регистре |
| **f** | Floating-point decimal |
| **F** | Floating-point decimal |
| **g** | Floating-point e или f |
| **C** | Floating-point E или F |
| **%** | Символьный % |

strip – удаляет пробелы слева и справа:

|  |  |
| --- | --- |
| 1  2 | print('       this is whitespace string    '.strip())  'this is whitespace string' |

translate – в отличие от replace, может делать множественную замену. В следующем примере каждый символ '1' в исходной строке будет заменен на символ '3', а символ '2' – на символ '4' соответственно:

|  |  |
| --- | --- |
| 1  2  3  4 | >>> from string import maketrans  >>> table = maketrans('12', '34')  >>> '1212 5656'.translate(table)  '3434 5656' |

Для конверсии различных типов в строковый используются функции str, int, ord, chr:

* str – конвертирует число в строку;
* int – конвертирует строку в число;
* ord – возвращает значение байта;
* chr – конвертирует число в символ.

##### **Таблица методов, доступных в Python 3.0**

| S.capitalize() S.ljust(width [, fill]) |
| --- |
| S.center(width [, fill]) S.lower() |
| S.count(sub [, start [, end]]) S.lstrip([chars]) |
| S.encode([encoding [,errors]]) S.maketrans(x[, y[, z]]) |
| S.endswith(suffix [, start [, end]]) S.partition(sep) |
| S.expandtabs([tabsize]) S.replace(old, new [, count]) |
| S.find(sub [, start [, end]]) S.rfind(sub [,start [,end]]) |
| S.format(fmtstr, \*args, \*\*kwargs) S.rindex(sub [, start [, end]]) |
| S.index(sub [, start [, end]]) S.rjust(width [, fill]) |
| S.isalnum() S.rpartition(sep) |
| S.isalpha() S.rsplit([sep[, maxsplit]]) |
| S.isdecimal() S.rstrip([chars]) |
| S.isdigit() S.split([sep [,maxsplit]]) |
| S.isidentifier() S.splitlines([keepends]) |
| S.islower() S.startswith(prefix [, start [, end]]) |
| S.isnumeric() S.strip([chars]) |
| S.isprintable() S.swapcase() |
| S.isspace() S.title() |
| S.istitle() S.translate(map) |
| S.isupper() S.upper() |
| S.join(iterable) S.zfill(width) |

## Далее подробнее.

## Таблица "Функции и методы строк"

| **Функция или метод** | **Назначение** |
| --- | --- |
| **S = 'str'; S = "str"; S = '''str'''; S = """str"""** | Литералы строк |
| **S = "s\np\ta\nbbb"** | Экранированные последовательности |
| **S = r"C:\temp\new"** | Неформатированные строки (подавляют экранирование) |
| **S = b"byte"** | Строка байтов |
| **S1 + S2** | Конкатенация (сложение строк) |
| **S1 \* 3** | Повторение строки |
| **S[i]** | Обращение по индексу |
| **S[i:j:step]** | Извлечение среза |
| **len**(S) | Длина строки |
| **S.find**(str, [start],[end]) | Поиск подстроки в строке. Возвращает номер первого вхождения или -1 |
| **S.rfind**(str, [start],[end]) | Поиск подстроки в строке. Возвращает номер последнего вхождения или -1 |
| **S.index**(str, [start],[end]) | Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError |
| **S.rindex**(str, [start],[end]) | Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError |
| **S.replace**(шаблон, замена) | Замена шаблона |
| **S.split**(символ) | Разбиение строки по разделителю |
| **S.isdigit**() | Состоит ли строка из цифр |
| **S.isalpha**() | Состоит ли строка из букв |
| **S.isalnum**() | Состоит ли строка из цифр или букв |
| **S.islower**() | Состоит ли строка из символов в нижнем регистре |
| **S.isupper**() | Состоит ли строка из символов в верхнем регистре |
| **S.isspace**() | Состоит ли строка из неотображаемых символов (пробел, символ перевода страницы ('\f'), "новая строка" ('\n'), "перевод каретки" ('\r'), "горизонтальная табуляция" ('\t') и "вертикальная табуляция" ('\v')) |
| **S.istitle**() | Начинаются ли слова в строке с заглавной буквы |
| **S.upper**() | Преобразование строки к верхнему регистру |
| **S.lower**() | Преобразование строки к нижнему регистру |
| **S.startswith**(str) | Начинается ли строка S с шаблона str |
| **S.endswith**(str) | Заканчивается ли строка S шаблоном str |
| **S.join**(список) | Сборка строки из списка с разделителем S |
| **ord**(символ) | Символ в его код ASCII |
| **chr**(число) | Код ASCII в символ |
| **S.capitalize**() | Переводит первый символ строки в верхний регистр, а все остальные в нижний |
| **S.center**(width, [fill]) | Возвращает отцентрованную строку, по краям которой стоит символ fill (пробел по умолчанию) |
| **S.count**(str, [start],[end]) | Возвращает количество непересекающихся вхождений подстроки в диапазоне [начало, конец] (0 и длина строки по умолчанию) |
| **S.expandtabs**([tabsize]) | Возвращает копию строки, в которой все символы табуляции заменяются одним или несколькими пробелами, в зависимости от текущего столбца. Если TabSize не указан, размер табуляции полагается равным 8 пробелам |
| **S.lstrip**([chars]) | Удаление пробельных символов в начале строки |
| **S.rstrip**([chars]) | Удаление пробельных символов в конце строки |
| **S.strip**([chars]) | Удаление пробельных символов в начале и в конце строки |
| **S.partition**(шаблон) | Возвращает кортеж, содержащий часть перед первым шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий саму строку, а затем две пустых строки |
| **S.rpartition**(sep) | Возвращает кортеж, содержащий часть перед последним шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий две пустых строки, а затем саму строку |
| **S.swapcase**() | Переводит символы нижнего регистра в верхний, а верхнего – в нижний |
| **S.title**() | Первую букву каждого слова переводит в верхний регистр, а все остальные в нижний |
| **S.zfill**(width) | Делает длину строки не меньшей width, по необходимости заполняя первые символы нулями |
| **S.ljust**(width, fillchar=" ") | Делает длину строки не меньшей width, по необходимости заполняя последние символы символом fillchar |
| **S.rjust**(width, fillchar=" ") | Делает длину строки не меньшей width, по необходимости заполняя первые символы символом fillchar |
| **S.format**(\*args, \*\*kwargs) | Форматирование строки |