



Строки в Python

литералы строк, операции со строками



Переменные

Простейшие типы

Строки

Операторы

Литералы строк

- В Python для объявления строк можно использовать одинарные ('), двойные (") и тройные (""" или ''')
- Для объявления многострочного текста используются **тройные** кавычки

```
1 single_quotes = 'some text'
2
3 double_quotes = "some text"
4
5 multiline_text = """
6 Hello, my friend!
7
8 I want to talk with you...
9 """
```



Вопрос-ответ

- ? Чем отличаются одинарные и двойные кавычки?
- ✓ Все одни обрабатываются одинаково. Идея в том, что вы можете создавать строки содержащие кавычки. Внутри одинарных можно расположить двойные и наоборот.

```
1 # Комбинирование кавычек
2 he_says = "I'm coming. Are you ready?"
3 she_says = 'She said "No, I need a couple of minutes"'
4
5 # Размещать одинарные кавычки внутри одинарных
6 # или двойные внутри двойных
7 # можно только дополнительно экранируя их
8 example = 'text\'text'
9 example2 = "text\"text"
10
11 # Следующие примеры вызовут ошибку SyntaxError
12 example3 = 'text'text'
13 example4 = "text"text"
```



Приводим другие типы к str

- Для того, чтобы привести другой тип данных к строчному, необходимо выполнить функцию `str(переменная_или_значение)`
- Для приведения к `str` вызывается метод класса `__str__`, а если он не определен, то `__repr__`

```
1 >>> str(True)
2 'True'
3 >>> str(False)
4 'False'
5 >>> str(123)
6 '123'
7 >>> str(1.0e4)
8 '10000.0'
9 >>> str((1, 2, 3))
10 '(1, 2, 3)'
11 >>> str({'hi': 'bye'})
12 "{'hi': 'bye'}"
13 >>> str([1, 2, 3])
14 '[1, 2, 3]'
```



Создаем управляющие последовательности с «\»

Управляющие последовательности позволяют с помощью некоторых сочетаний добиться эффекта, который по-другому было бы трудно выразить

- `\n` – перевод строки
- `\t` – горизонтальная табуляция
- `\a` – звонок
- `\\` – экранировать символ «\» в строке
- `\'` – экранировать символ «'» в строке
- `\"` – экранировать символ «"» в строке



Объединяем строки (+, конкатенация)

- С помощью символа «+» можно объединять строки или строковые переменные
- Python **не добавляет** пробелы за вас

```
1 >>> 'Hello ' + 'World' + '!'
2 'Hello World!'
3 >>> a = 'Hello'
4 >>> b = 'World'
5 >>> a + ' ' + b
6 'Hello World'
```



Извлекаем символ из строки ([])

- Для того, чтобы извлечь символ из строки, необходимо воспользоваться квадратными скобками
- Нумерация символов в строке **начинается с нуля**
- Можно извлекать символы с конца строки, используя отрицательный индекс, например `some_string[-3]`

```
1 surname = 'Ринейский'
2 name = 'Вячеслав'
3 patronymic = 'Викторович'
4 fio = f"{surname[0]}.{name[0]}.{patronymic[0]}."
5 print(fio)
```

Run: example x

D:\SLAVA\projects\bh_2_classwork\venv\Scripts\p
P.B.B.
|
Process finished with exit code 0



Строки – неизменяемый тип данных

Поэтому нельзя вставить в строку символ на определенную позицию

```
1 >>> name = 'Terry'
2 >>> name[0] = 'P'
3 Traceback (most recent call last):
4   File "<stdin>", line 1, in <module>
5   TypeError: 'str' object does not support item assignment
```



Извлекаем срез строки с помощью [start:end:step]

Для извлечения среза (некоторой части) строки используется оператор [start:end:step]:

- [:] – получаем всю строку
- [start:] – последовательность со start до конца строки
- [:end] – последовательность от начала до (end – 1)
- [start:end] – последовательность от start до (end – 1)
- [start:end:step] – последовательность от start до (end – 1) из символов, чье смещение кратно step

```
1 >>> string = "Bon Jovi - Livin' On A Prayer"
2 >>> string[:]
3 "Bon Jovi - Livin' On A Prayer"
4 >>> string[11:]
5 "Livin' On A Prayer"
6 >>> string[:8]
7 'Bon Jovi'
8 >>> string[-6::2]
9 'Pae'
10 >>> string[11:16:3]
11 'Li'
```



Получаем длину строки (len)

- Для того, чтобы получить длину строки, необходимо воспользоваться функцией `len(строка)`
- Результат часто используется в условиях

```
1 >>> nickname = 'Bloha_V_SaRaFFane'
2 >>> if len(nickname) > 15:
3 ...     print('Sorry, this nickname is too long')
4 ...
5 Sorry, this nickname is too long
6 >>>
```



Raw-строки подавляют экранирование

Бывают случаи, когда слеш-последовательности могут сыграть злую шутку. Для подавления механизма экранирования, нужно перед строкой добавить символ «r»

Но, несмотря на назначение, raw-строка не может заканчиваться символом обратного слэша. Пути решения:

- Прибавлять к raw-строке что-то со слешем
- Экранировать слеш и не использовать raw-строку

```
1 path = 'C:\new folder\txt documents.txt'
2 print(path)
3
4 raw_path = r'C:\new folder\txt documents.txt'
5 print(raw_path)
```

Run: example (1) x

D:\SLAVA\projects\bh_2_classwork\venv\Script
C:
ew folder xt documents.txt
C:\new folder\txt documents.txt

Process finished with exit code 0



Способы форматирования строк

Конкатенация. Грубый способ форматирования, в котором мы просто склеиваем строки с помощью операции сложения

- Плохо читается
- Не всегда с ним удобно работать

```
1  >>> numb = 123
2  >>> word_1 = 'Hello'
3  >>> word_2 = 'World'
4  >>> word_1 + ' ' + str(numb) + ' ' + word_2
5  'Hello 123 World'
```



Способы форматирования строк

%-форматирование. Способ, который перешел из языка C.

- Можно передавать значения через списки, кортежи и словари

```
1 >>> name = "Дмитрий"
2 >>> age = 38
3 >>> "Меня зовут %s. Мне %d лет." % (name, age)
4 'Меня зовут Дмитрий. Мне 38 лет.'
5 >>> "Меня зовут %(name)s. Мне %(age)d лет." % {"name": name, "age": age}
6 'Меня зовут Дмитрий. Мне 38 лет.'
```



Способы форматирования строк

Template-строки. Этот способ появился в Python 2.4, как замена %-форматированию (PEP 292), но популярным так и не стал.

- Поддерживает передачу значений по имени и использует \$-синтаксис как в PHP
- Не рекомендуется к использованию

```
1 >>> from string import Template
2 >>> name = "Дмитрий"
3 >>> age = 38
4 >>> s = Template('Меня зовут $name. Мне $age лет.')
5 >>> s.substitute(name=name, age=age)
6 'Меня зовут Дмитрий. Мне 38 лет.'
```



Способы форматирования строк

Форматирование с помощью метода `format()`. Этот способ появился в Python 3 в качестве замены %-форматированию.

- Он также поддерживает передачу значений по позиции и по имени

```
1 >>> name = "Дмитрий"
2 >>> age = 38
3 >>> 'Меня зовут {}. Мне {} лет.'.format(name, age)
4 'Меня зовут Дмитрий. Мне 38 лет.'
5 >>> 'Меня зовут {name}. Мне {age} лет.'.format(name=name, age=age)
6 'Меня зовут Дмитрий. Мне 38 лет.'
```



Способы форматирования строк

f-строки. Форматирование, которое появилось в Python 3.6 (PEP 498). Этот способ похож на форматирование с помощью метода `format()`, но гибче, читабельней и быстрее.

```
1 >>> name = 'Дмитрий'
2 >>> age = 38
3 >>> f"Меня зовут {name}. Мне {age} лет."
4 'Меня зовут Дмитрий. Мне 38 лет.'
```



Как форматировать вывод

Поле замены имеет вид:

- `"{[поле][!преобразование][:спецификация]}"`

Например: `f"{dogs.get_all()[0].height:.2f}"`

```
1 >>> f_list = [1.124, 1.025]
2 >>> f"0 элемент: {f_list[0]:+.2f}"
3 '0 элемент: +1.12'
4 >>> f"1 элемент: {f_list[1]:^20}"
5 '1 элемент:           1.025'
```



Поле

Имеет вид:

- имя[.имя_атрибута | .имя_метода(параметры) | [индекс]]

Для f-строк имя – имя переменной.

Для format имя :

- Номер элемента, если передаем значения через запятую
- Имя поля, если передаем значения через словарь



Преобразование

- !r – результат функции repr()
- !s – результат функции str()
- !a – результат функции ascii()

```
1  a = "Harold's a clever {0!s}"      # Calls str()
2  b = "Bring out the holy {name!r}"  # Calls repr()
3  c = "More {!a}"                   # Calls ascii()
```



Спецификация

Имеет вид:

- [[заполнение] выравнивание][знак][#][0][ширина][символ группировки][.точность][тип]



Заполнение и выравнивание

Выравнивание производится при помощи символа-заполнителя. Доступны следующие варианты выравнивания:

- < - Символы-заполнители будут справа (выравнивание объекта по левому краю) (по умолчанию)
- > - выравнивание объекта по правому краю
- = - Заполнитель будет после знака, но перед цифрами. Работает только с числовыми типами
- ^ - Выравнивание по центру

```
1 >>> f"{'hello':*^20}"
2 '*****heLlo*****'
```



Знак

Опция "знак" используется только для чисел и может принимать следующие значения:

- `+`: знак должен быть использован для всех чисел
- `-`: `'-'` для отрицательных, ничего для положительных
- `'Пробел'`: `'-'` для отрицательных, пробел для положительных

```
1  >>> f"{1000: }"  
2  ' 1000'  
3  >>> f"{1000:-}"  
4  '1000'  
5  >>> f"{-1000:-}"  
6  '-1000'  
7  >>> f"{-1000:+}"  
8  '-1000'  
9  >>> f"{1000:+}"  
10 '+1000'
```



#, 0, ширина

- альтернативная форма представления

0 – дополнение нулями

```
1 >>> f"{1000:b}"
2 '1111101000'
3 >>> f"{1000:#b}"
4 '0b1111101000'
5 >>> f"{1000:010}"
6 '0000001000'
7 >>> f"{1000:05}"
8 '01000'
```



Символ группировки (grouping option)

Для int и float разбивает число по 10^{**3} . Возможные значения:

- _
- ,

```
1 >>> f"{100000:_}"
2 '100_000'
3 >>> f"{1000000:_}"
4 '1_000_000'
5 >>> f"{1000000:,}"
6 '1,000,000'
```



Точность

Количество знаков после запятой:

- .[количество]
- .2
- .4

```
1 >>> f"{10.123321:.2f}"  
2 '10.12'
```



Тип

'd', 'i', 'u'	Десятичное число
'o'	Число в восьмеричной системе счисления
'x'	Число в шестнадцатеричной системе счисления (буквы в нижнем регистре)
'X'	Число в шестнадцатеричной системе счисления (буквы в верхнем регистре)
'e'	Число с плавающей точкой с экспонентой (экспонента в нижнем регистре)
'E'	Число с плавающей точкой с экспонентой (экспонента в верхнем регистре)
'f', 'F'	Число с плавающей точкой (обычный формат)
'g'	Число с плавающей точкой. с экспонентой (экспонента в нижнем регистре), если она меньше, чем -4 или точности, иначе обычный формат
'G'	Число с плавающей точкой. с экспонентой (экспонента в верхнем регистре), если она меньше, чем -4 или точности, иначе обычный формат
'c'	Символ (строка из одного символа или число - код символа)
's'	Строка
'%'	Число умножается на 100, отображается число с плавающей точкой, а за ним знак %



Разбиваем строку по разделителю (split)

- Чтобы разбить строку по разделителю – используем функцию `строка.split(разделитель)`.
- Результат – `list[str]`
- Например, у нас есть рецепт блюда, где ингредиенты представлены через запятую, то мы можем разбить данную строку с разделителем запятая
- Используется, например, когда данные в БД сохранены специфичным образом через разделитель

```
1 >>> numbers = 'one,two,three'
2 >>> numbers.split(',')
3 ['one', 'two', 'three']
```



Объединяем строки через разделитель (join)

- Для того, чтобы объединить строки из некоторого списка в одну используется функция join (по сути, обратная функция для split)
- Обращаем внимание на использование: 'Разделитель'.join(['а', 'б'])

```
1 >>> great_evening = ['cola', 'pizza', 'film']
2 >>> f"Great evening is {'', '.join(great_evening)}"
3 'Great evening is cola, pizza, film'
```



Поиск подстроки в строке

- `S.find(str, [start],[end])`. Возвращает номер первого вхождения или -1
- `S.rfind(str, [start],[end])`. Возвращает номер последнего вхождения или -1
- `S.index(str, [start],[end])`. Возвращает номер первого вхождения или вызывает `ValueError`
- `S.rindex(str, [start],[end])`. Возвращает номер последнего вхождения или вызывает `ValueError`
- `S.count(str, [start],[end])`. Возвращает количество непересекающихся вхождений подстроки в диапазоне [начало, конец] (0 и длина строки по умолчанию)
- `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

```
1 >>> 'Hello World'.find('or')
2 7
3 >>> 'Hello World'.rfind('or')
4 7
5 >>> 'Hello World'.find('o')
6 4
7 >>> 'Hello World'.rfind('o')
8 7
9 >>> 'Hello World'.index('o')
10 4
11 >>> 'Hello World'.rindex('o')
12 7
13 >>> 'Hello World'.rindex('z')
14 Traceback (most recent call last):
15   File "<stdin>", line 1, in <module>
16 ValueError: substring not found
17 >>> 'Hello World'.count('l')
18 3
19 >>> 'Hello World'.partition('l')
20 ('He', 'l', 'lo World')
21 >>> 'Hello World'.rpartition('l')
22 ('Hello Wor', 'l', 'd')
```



Замена шаблона

- `S.replace(шаблон, замена)`

```
1 >>> 'Hello World'.replace('Hello', 'Goodbye')
2 'Goodbye World'
```



Состоит ли строка из

- `S.isdigit()`. Состоит ли строка из цифр
- `S.isalpha()`. Состоит ли строка из букв
- `S.isalnum()`. Состоит ли строка из цифр или букв
- `S.isidentifier()`. Является ли строка валидным идентификатором
- `S.isspace()`. Состоит ли строка из неотображаемых символов
- `S.startswith(str)`. Начинается ли строка `S` с шаблона `str`
- `S.endswith(str)`. Заканчивается ли строка `S` шаблоном `str`

```
1 >>> 'Hello World'.isdigit()
2 False
3 >>> 'Hello World'.isalpha()
4 False
5 >>> 'HelloWorld'.isalpha()
6 True
7 >>> 'HelloWorld'.isalnum()
8 True
9 >>> 'Hello World'.isalnum()
10 False
11 >>> 'Hello World'.isspace()
12 False
13 >>> 'Hello World'.startswith('Hello')
14 True
15 >>> 'Hello World'.startswith('Bye')
16 False
17 >>> 'Hello World'.endswith('World')
18 True
```



Регистр

- `S.islower()`. Состоит ли строка из символов в нижнем регистре
- `S.isupper()`. Состоит ли строка из символов в верхнем регистре
- `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.capitalize()`. Переводит первый символ строки в верхний регистр, а все остальные в нижний
- `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

```
1 >>> 'Hello World'.islower()
2 False
3 >>> 'Hello World'.isupper()
4 False
5 >>> 'Hello World'.istitle()
6 True
7 >>> 'Hello World'.upper()
8 'HELLO WORLD'
9 >>> 'Hello World'.lower()
10 'hello world'
11 >>> 'Hello World'.capitalize()
12 'Hello world'
13 >>> 'Hello World'.swapcase()
14 'hELLO wORLD'
15 >>> 'hello world'.title()
16 'Hello World'
```



Обработка строки

- `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
- `str.removeprefix(prefix)`. С Python 3.9 Удаляет шаблон с начала строки
- `str.removesuffix(suffix)`. С Python 3.9 удаляет шаблон с конца строки

```
1 >>> '\tHello World'.expandtabs()
2 '      Hello World'
3 >>> '      Hello World'.lstrip()
4 'Hello World'
5 >>> 'Hello World      '.rstrip()
6 'Hello World'
7 >>> '      Hello World      '.strip()
8 'Hello World'
9 >>> 'Hello World'.removeprefix('Hel')
10 'lo World'
11 >>> 'Hello World'.removesuffix('ld')
12 'Hello Wor'
```



Дополнение строки

- `S.center(width, [fill])`. Возвращает отцентрированную строку, по краям которой стоит символ `fill` (пробел по умолчанию)
- `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`

```
1 >>> 'Hello World'.center(20, '-')
2 '----Hello World-----'
3 >>> 'Hello World'.zfill(20)
4 '00000000Hello World'
5 >>> 'Hello World'.ljust(20, '-')
6 'Hello World-----'
7 >>> 'Hello World'.rjust(20, '-')
8 '-----Hello World'
```



Кодировки строк

Кодировка текста — это способ перевода символов (таких как буквы, знаки пунктуации, служебные знаки, пробелы и контрольные символы) в целые числа и затем непосредственно в биты.

В Python 3 по умолчанию используется кодировка UTF-8



ASCII

ASCII — название таблицы, в которой некоторым распространённым печатным и непечатным символам сопоставлены числовые коды. Таблица была разработана и стандартизирована в США, в 1963 году

NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
SPC	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
Ђ	Ѓ	Ѕ	Ї	Ї	Ї	Ї	Ї	Ї	Ї	Ї	Ї	Ї	Ї	Ї	Ї
ђ	ѓ	ѕ	ї	ї	ї	ї	ї	ї	ї	ї	ї	ї	ї	ї	ї
Ў	ў	Ј	ј	Љ	љ	Ћ	ћ	Ќ	ќ	Ў	ў	Ј	ј	Љ	љ
°	±	І	і	Г	г	М	м	Ј	ј	Љ	љ	Ћ	ћ	Ќ	ќ
А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

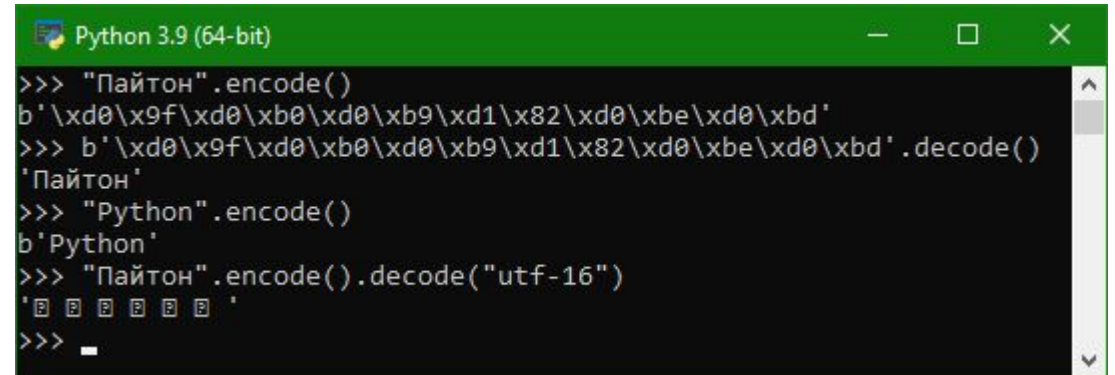


Перевод в байты и обратно

В python для перевода строки в байтстроку и обратно используются функции

- `str.encode()`
- `str.decode()`

Если знак относится к ASCII, то его байтовое представление будет выглядеть как оригинальный символ. В случае когда он выходит за пределы ASCII, то заменяется байтовым представлением (`\x` - эскейп-последовательность для обозначения 16-ричных чисел в языке Python)



```
Python 3.9 (64-bit)
>>> "Пайтон".encode()
b'\xd0\x9f\xd0\xb0\xd0\xb9\xd1\x82\xd0\xbe\xd0xbd'
>>> b'\xd0\x9f\xd0\xb0\xd0\xb9\xd1\x82\xd0\xbe\xd0xbd'.decode()
'Пайтон'
>>> "Python".encode()
b'Python'
>>> "Пайтон".encode().decode("utf-16")
'\x00\x00\x00\x00\x00\x00'
```

