Строки

Строки в Python — это последовательности. Элементы этих последовательностей — текстовые символы. Обработка, отображение или хранение текстовой информации — все эти операции выполняются посредством строк.

Разработчику нужно уметь работать со строками, потому что строки — это текст, и работа с ними позволяет обрабатывать и изменять текстовую информацию в программе. Это полезно для создания текстовых приложений и обработки текстовых данных.

Синтаксис строк

Обычно строки замыкают в двойные или одинарные кавычки:

```
txt = 'Я строка, во мне есть смысл, а Python видит во мне лишь набор символов.'
txt = "Я тоже строка! Но для Python я просто последовательность. Печаль."
```

Если для оформления строки применяются одинарные кавычки, то вложенные должны быть двойными:

```
error_string = 'Сопряжение с устройством "трекер Unicorn" прервано.'
```

Объявление строк

Объявление строки в одинарных кавычках

Длинные строки неудобно читать в коде, PEP8 не рекомендует делать их длиннее 79 символов.

Перенос строк (значений типа str) в коде делается так:

- вся строка целиком замыкается в скобки;
- текст внутри скобок разбивается построчно, на фрагменты нужной длины;
- пробелы (если они есть) оставляют в конце строчки: строчки не начинают с пробела;
- каждая получившаяся строчка замыкается в кавычки;
- для читаемости все строчки отбиваются пробелами от начала строки.

```
alert_string = ('Потребление калорий превысило расход калорий на 300%! '
                'Уберите бургер в холодильник!')
```

Объявление строки в тройных кавычках

Этот синтаксис чаще используют для описания функций или классов, для создания многострочных комментариев или для подстановки длинного SQL-запроса в тело программы.

```
output_string = '''Количество шагов за день 18500
Пройденная дистанция 12 км.
Отличный рузультат!'''
```

Объявление строки функцией str()

```
monday_steps = 12000  # Это число
tuesday_steps = 9000  # Это число
# Сложение будет проведено по правилам арифметики
print(monday_steps + tuesday_steps)
# Вывод в терминал: 21000

# Функция str() преобразует переданное ей значение в строку (если такое возможно)
monday_steps_str = str(monday_steps)  # В monday_steps_str передана строка '12000'
tuesday_steps_str = str(tuesday_steps) # В tuesday_steps_str передана строка 9000'

# Сложение пойдёт по иным правилам: последовательности символов просто "склеятся"
print(monday_steps_str + tuesday_steps_str)
# Вывод в терминал: 120009000
# Это уже не арифметика, а конкатенация какая-то!
```

Методы строк

string.lower() и string.upper()

Эти методы возвращают копию строки, в которой все символы переведены, соответственно, в нижний или в верхний регистр.

```
status_string = 'ВСЁ ИДЁТ ПО ПЛАНУ'
# Если всё по плану – можно написать в нижнем регистре:
print(status_string.lower())
# Вывод в терминал: всё идёт по плану

alarm_string = 'всё сломалось, проект не работает!'
# Переводим текст сообщения в верхний регистр
print(alarm_string.upper())
# Вывод в терминал: ВСЁ СЛОМАЛОСЬ, ПРОЕКТ НЕ РАБОТАЕТ!
```

string.capitalize()

Возвращает копию строки, в которой первый символ переведён в верхний регистр, а остальные — в нижний.

```
distance_string = 'сегодня ВЫ прошли 12 КМ'
print(distance_string.capitalize())
# Вывод в терминал: Сегодня вы прошли 12 км
```

string.title()

Возвращает копию строки, в которой первый символ каждого слова переведён в верхний регистр, а остальные — в нижний (заголовки в английском языке пишут именно в таком формате).

```
user_name_string = 'пользователь антон'
print(user_name_string.title())
# Вывод в терминал: Пользователь Антон
```

string.swapcase()

Возвращает копию строки, в которой регистр символов инвертирован.

```
inverted_string = 'ПрОйДеНо ШаГоВ зА ДеНь: 18500'
print(inverted_string.swapcase())
# Вывод в терминал: пРоЙдЕнО шАгОв За дЕнЬ: 18500
```

string.find()

Возвращает индекс первого символа в строке, где была найдена нужная подстрока.

```
spent_calories = 'Сегодня вы сожгли 500 калорий'
print(spent_calories.find('500'))
# Вывод в терминал: 18
```

string.replace(src, new)

Возвращает копию строки, заменяя подстроку src на подстроку new.

```
cat_motto = 'Котики важны. Котики улучшают эмоциональное здоровье.'

fitness_motto = cat_motto.replace('Котики', 'Тренировки')
print(fitness_motto)
# Вывод в терминал: Тренировки важны. Тренировки улучшают эмоциональное здоровье.

# Третьим параметром можно указать количество заменяемых подстрок.
fitness_motto = cat_motto.replace('Котики', 'Тренировки', 1)
print(fitness_motto)
# Вывод в терминал: Тренировки важны. Котики улучшают эмоциональное здоровье.
```

string.strip()

Возвращает копию строки, удаляя из её начала и конца все символы, указанные как аргумент. Эти символы могут передаваться в аргумент в произвольном порядке.

```
error_message = 'ER03:"Ошибка подключения устройства! Доступ блокирован!"'
text = error_message.strip('"E3!0R:')
print(text)
# Вывод в терминал: Ошибка подключения устройства! Доступ блокирован
# Символы ER03 в начале строки удалены, хотя были переданы в strip() в другом по-
рядке.
# Двойная кавычка удалена в начале и в конце строки.
# Восклицательный знак в конце строки удалён, но в середине — сохранился.
```

string.split()

Разбивает заданную строку на части и возвращает список этих частей. Символ, по которому строка должна быть разделена, указывается в необязательном аргументе метода. Если аргумент не указан — разделителем будет считаться символ пробела; в результате строка будет разбита на слова.

```
workout = 'Бег; Подтягивания; Отжимания; Приседания; Полежать'
# Разбиваем строку, не указав аргумент для split()
print(workout.split())
# Выведется: ['Бег;', 'Подтягивания;', 'Отжимания;', 'Присядания;', 'Полежать']
```

В качестве разделителя можно передать любой символ; в результате исходная строка будет разбита по нему. Разделитель должен быть строкой.

```
workout = 'Бег; Подтягивания; Отжимания; Приседания; Полежать'
# Разделитель — точка с запятой
print(workout.split(';'))
# Выведется: ['Бег', ' Подтягивания', ' Отжимания', ' Приседания', ' Полежать']

# Разделитель не обязательно должен состоять из одного символа:
print(workout.split('ан'))
# Выведется: ['Бег; Подтягив', 'ия; Отжим', 'ия; Присед', 'ия; Полежать']
```

delimiter.join(list)

Метод, обратный предыдущему: он возвращает строку, составленную из списка строк; между фрагментами созданной строки будет поставлен разделитель (он должен быть строкой, как и в split()).

```
workout = ['Бег', 'Подтягивания', 'Отжимания', 'Приседания', 'Полежать']
print('; '.join(workout))  # Разделителем будет точка с запятой и пробел
# После последнего фрагмента разделитель не ставится.
# Вывод в терминал: Бег; Подтягивания; Отжимания; Приседания; Полежать
```

string.format(аргумент)

Этот метод позволяет подставлять в строку значения переменных. Значение переменной вставляется на место фигурных скобок в строке.

```
steps = 10564
print('Вы прошли за день {} шага.'.format(steps))
# Вывод в терминал: Вы прошли за день 10564 шага.
```

f-строки

f-string (от англ. formatted string — «отформатированная строка»).

f-строки задаются с помощью литерала f перед кавычками, обрамляющими строку:

```
distance = 10.564
fstring = f'За день вы прошли {distance} км'

print(fstring)
# Вывод в терминал: За день вы прошли 10.564 км
```

В f-строках доступно и расширенное форматирование значений, и основные арифметические операции:

```
dist_km = 0.539
print(f'За день вы прошли {dist_km:.2f} км')
# Вывод в терминал: За день вы прошли: 0.54 км

achievement_part = 0.657589
# Переобразовать выводимое число в проценты,
# округлив до второго знака после запятой
print(f'Прогресс достижения цели: {achievement_part:.2%}')
# Вывод в терминал: Прогресс достижения цели: 65.76%

# Посчитаем:
steps = 8463 # Количество пройденных шагов
len_step_m = 0.65 # Длина одного шага в метрах
transfer_coeff = 1000 # Метров в километре

print(f'За день вы прошли {steps * len_step_m / transfer_coeff:.2f} км')
# Вывод в терминал: За день вы прошли 5.50 км
```

В f-строках можно вызывать методы, доступные для переданной переменной. Например, можно вызвать метод upper() для переданной переменной типа строка:

```
name = 'Антон'
print(f'Данные пользователя {name.upper()} загружены.')
# Вывод в терминал: Данные пользователя АНТОН загружены.
```

Полезные ресурсы

Рекомендации по оформлению кода Python: PEP8 (Python Enhancement Proposals)

Раздел документации о методе string.format

Раздел документации об f-строках