

ХОЧУ ПОМОЧЬ
ПРОЕКТУ

Текстовые константы и шаблоны, модуль string

Практикуйтесь

practicum.yandex.ru

Профессия: Python-разработчик. Курс от Яндекса.
Поможем освоить новую профессию с нуля за 9 месяцев. Начните учиться бесплатно!
[Узнать больше](#)

otus.ru

Собираетесь стать Linux-администратором или DevOps?

Подготовительный онлайн-курс от OTUS, разберем основы администрирования за 13 тем

[Узнать больше](#)

В большинстве случаев встроенный метод `str.format()` [класса `str`](#) представляет собой более удобный интерфейс для функций форматирования переменных, но класс `Formatter` модуля `string` предоставляется способ создания подклассов для случаев, когда требуются изменения в способе форматирования.

Шаблонные строки.

[Шаблоны строк](#) предназначены в качестве альтернативы встроенному синтаксису замены переменных при [форматировании строк](#). При интерполяции строки по шаблону `string.Template` переменные идентифицируются с помощью префикса `$`, например, `$var`. В качестве альтернативы, если необходимо выделить их из окружающего текста, они также могут быть обернуты фигурными скобками, например `${var}`.

```
import string

values = {'one': 'Привет', 'two': 'коп'}

t = string.Template("""
Просто переменная: $one
Экранируем префикс: $$
Переменная в тексте: квадро${two}тер.
""")

print(t.substitute(values))
# Просто переменная: Привет
# Экранируем знак доллара: $
# Переменная в тексте: квадрокоптер.
```

Ключевым отличием [шаблонов](#) от строковых подстановок (интерполяции) и форматирования с помощью [метода `format\(\)`](#) является то, что тип аргументов не принимается во внимание. Значения преобразуются в строки, а строки вставляются в результат. В шаблонах недоступны параметры форматирования переменных. Например, невозможно контролировать количество цифр, используемых для представления значения с плавающей запятой.

Преимуществом является то, что использование [метода `safe_substitute\(\)`](#) позволяет избежать [исключений](#), если не все значения, необходимые шаблону, предоставляются в качестве аргументов.

```
import string

values = {'var': 'foo'}

t = string.Template("$var is here but $missing is not provided")
try:
    print('substitute():', t.substitute(values))
except KeyError as err:
    print('ERROR:', str(err))

print('safe_substitute():', t.safe_substitute(values))
```

Вверх

```
# ERROR: 'missing'
# safe_substitute(): foo is here but $missing is not provided
```

Так как [в словаре](#) values отсутствует значение для переменной шаблона \$missing, то вызывается [исключение KeyError](#) при использовании метода шаблона [substitute\(\)](#). Метод safe_substitute(), вместо того, чтобы вызвать ошибку ловит ее и вставляет в текст.

Изменение шаблонов.

[string.Template](#) можно изменить, заменив их шаблонами на основе [регулярных выражений](#), которые определяют имена переменных в теле шаблона. Простой способ сделать это-изменить разделитель delimiter

🚩 otus.ru

Собираетесь стать Linux-администратором или DevOps?

Подготовительный онлайн-курс от OTUS, разберем основы администрирования за 13 тем

Узнать больше

```
d = {'with_underscore': 'replaced', 'notunderscored': 'not replaced'}

t = MyTemplate(template_text)
print(t.safe_substitute(d))
# Delimiter : %
# Replaced  : replaced
# Ignored   : %notunderscored
```

В примере правила подстановки изменены таким образом, что разделителем является % вместо \$, а имена переменных должны содержать символ подчеркивания. Из примера видно, что шаблон %notunderscored - без подчеркивания ничем не заменяется.

Для **еще более сложных изменений** можно переопределить атрибут pattern и задать совершенно новое регулярное выражение. Атрибут pattern должен содержать четыре именованные группы: escaped - для захвата экранированного разделителя, именованную переменную named, фигурную версию имени переменной braced и недопустимые шаблоны разделителя invalid. (Подробнее смотрите описание [класса string.Template\(template\)](#))

Стандартный атрибут pattern можно получить следующим способом:

```
import string

t = string.Template('$var')
print(t.pattern.pattern)
# \${(?:
#   (?P<escaped>\$) |
#   (?P<named>[_a-z][_a-z0-9]*) |
#   {(?P<braced>[_a-z][_a-z0-9]*)} |
#   (?P<invalid>)
# )
```

Определим новый шаблон для использования {{var}} в качестве синтаксиса переменной.

```
import re
import string

# Переопределяем delimiter и pattern
class NewTemplate(string.Template):
    delimiter = '{{{'
    pattern = r'''
    \{\{\(?:
    (?P<escaped>\{\}) |
    (?P<named>[_a-z][_a-z0-9]*)\}\} |
    (?P<braced>[_a-z][_a-z0-9]*)\}\} |
    (?P<invalid>)
```

Вверх

```
t = NewTemplate('Привет {{var}}!')
print(t.safe_substitute(var='Мир'))
# Привет Мир!
```



otus.ru

Собираетесь стать Linux-администратором или DevOps?

Подготовительный онлайн-курс от OTUS разберем основы DOCS-Python.ru™, 2023 г.

Узнать больше

Содержание раздела:

ring

string.Template

(Внимание! При копировании материала ссылка на источник обязательна)

@docs_python_ru