******

**Инструкция по использованию и**

**форматированию строк**

**Форматирования строк “По старому”.**

[Строки в Python](https://python-scripts.com/string) содержат уникальную встроенную операцию, доступ к которой можно получить через **оператор %**. Это позволяет заметно упростить позиционное форматирование. Если вы когда-либо работали с функцией **printf** в С, вы сразу узнаете, как это работает. Вот простой пример:

print('Hello, %s' % name)

# Вывод: "Hello, Bob"

Я использую определитель формата **%s** в данном случае, чтобы сказать Python, где именно заменить значение имени, представленного в виде строки.

Существуют другие определители формата, которые позволяют вам контролировать формат выдачи. Например, возможно **конвертировать числа** в шестнадцатеричную нотацию или добавлять пробелы для создания хорошо отформатированных таблиц и отчетов.

Здесь, вы можете использовать определитель формата %x для конвертации значения int в строку и представить его в качестве шестнадцатеричного числа:

errno = 50159747054

print('%x' % errno)

# Вывод: 'badc0ffee'

“По старинке”, синтаксис форматирования строки немного меняется, если вы хотите сделать несколько замен в одной строке. Так как оператор **%** принимает только один аргумент, вам нужно обернуть правую часть в [кортеж](https://python-scripts.com/lists-tuples-dictionaries#tuple), вот так:

errno = 50159747054

name = 'Bob'

print('Hey %s, there is a 0x%x error!' % (name, errno))

# 'Hey Bob, there is a 0xbadc0ffee error!'

Также возможно сослаться на заменители переменных по имени в вашей строке формата, если вы передадите сопоставление **оператору %**:

print(

'Hey %(name)s, there is a 0x%(errno)x error!' % {

"name": name, "errno": errno

}

)

# Вывод: 'Hey Bob, there is a 0xbadc0ffee error!'

Это помогает сделать настройки вашего формата более простым процессом, а также упрощает его модификацию в будущем. Вам не нужно беспокоиться о том, что порядок, который вы передаете в значениях, совпадает с порядком, в котором значения указаны в строке формата. Конечно, недостаток — это то, что эта техника требует немного больше ручного ввода.

Я уверен, что вы думаете, почему это **printf** форматирование считается**старым форматированием строк**. Технически, оно было заменено новым подходом к форматированию в “[Python 3](http://python-3.ru/" \t "_blank)”, которое мы сейчас и рассмотрим!

[**2 Форматирование строк “По новому” (str.format)**](https://python-scripts.com/string-formatting#str-format)**.**

**Python 3** предоставил новый способ форматирования, который также был внесен в раннюю версию **Python 2.7**. Этот “новый стиль” форматирования строк избавляется от специального синтаксиса **оператора %** и делает синтаксис для форматирования строк более регулярным. Теперь форматирование обрабатывается вызовом .**format**() в объекте строки.

Вы можете использовать **format**(), чтобы выполнить простое позиционное форматирование, также, как мы делали это по старинке:

print('Hello, {}'.format(name)) # Вывод: 'Hello, Bob'

print('Hey {name}, there is a 0x{errno:x} error!'.format(name=name, errno=errno)

# Вывод: 'Hey Bob, there is a 0xbadc0ffee error!'

**3 Интерполяция строк / f-Строки (Python 3.6+)**

Python 3.6 Добавил новый подход форматирования строк под названием форматированные строчные литералы, или “f-строки”. Этот новый способ форматирования строк позволяет вам использовать встроенные выражения Python внутрь строковых констант. Вот простой, наглядный пример:

name = 'Bob'

print(f'Hello, {name}!')

# Вывод: 'Hello, Bob!'

Как вы видите, это добавляет префикс к константе строки с буквой “f” — следовательно, названием становится “f-strings”. Этот новый синтаксис форматирования — очень мощный. Так как вы можете вставлять произвольные выражения Python, вы можете даже проводить встроенную арифметику. Посмотрим на пример:

a = 5

b = 10

print(f'Five plus ten is {a + b} and not {2 \* (a + b)}.')

# Вывод: 'Five plus ten is 15 and not 30.'

Форматированные строчные литералы — это особенность парсера Python, которая конвертирует f-строки в серию строчных констант и выражений. Затем, они соединяются и составляют итоговую строку.

Представьте, что у вас есть следующая функция greet(), которая содержит f-строку:

def greet(name, question):

return f"Hello, {name}! How's it {question}?"

print(greet('Bob', 'going'))

# Вывод: "Hello, Bob! How's it going?"

Когда вы разбираете функцию, и смотрите, что происходит за кулисами, вы увидите, что **f-строка в функции** трансформируется в нечто, похожее на следующее:

def greet(name, question):

     return "Hello, " + name + "! How's it " + question + "?"

Настоящая имплементация проходит немного быстрее чем в примере, так как использует опкод BUILD\_STRING в качестве оптимизации, однако с точки зрения функционала они одинаковы:

>>> import dis

>>> dis.dis(greet)

  2           0 LOAD\_CONST               1 ('Hello, ')

              2 LOAD\_FAST                0 (name)

              4 FORMAT\_VALUE             0

              6 LOAD\_CONST               2 ("! How's it ")

              8 LOAD\_FAST                1 (question)

             10 FORMAT\_VALUE             0

             12 LOAD\_CONST               3 ('?')

             14 BUILD\_STRING             5

             16 RETURN\_VALUE

**Строчные литералы** также поддерживают существующий синтаксис формата строк метода **str.format**(). Это позволяет вам решать те же проблемы с форматированием, которые мы рассматривали в двух предыдущих разделах:

print(f"Hey {name}, there's a {errno:#x} error!")

# Вывод: "Hey Bob, there's a 0xbadc0ffee error!"

**4 Шаблонные строки (Стандартная библиотека Template Strings)**

Рассмотрим еще один инструмент для форматирования строк в Python: **template strings**. Это более простой и менее мощный механизм, но в ряде случаев он может быть именно тем, что вам нужно.

Давайте посмотрим на простой пример:

from string import Template

t = Template('Hey, $name!')

print(t.substitute(name=name))

# Вывод: 'Hey, Bob!'

Вы видите, что нам нужно импортировать класс Template из встроенного модуля Python, под названием string. Эти шаблонные строки не являются особенностью корневого языка, но они поддерживаются модулем string в стандартной библиотеке.

Другое отличие заключается в том, что шаблонные строки не позволяют форматировать спецификаторы. Учитывая это, чтобы сделать предыдущий пример с ошибкой рабочим, вам нужно вручную изменить номер ошибки int в шестнадцатериPython

templ\_string = 'Hey $name, there is a $error error!'

print(Template(templ\_string).substitute(name=name, error=hex(errno)))

# Вывод: 'Hey Bob, there is a 0xbadc0ffee error!'

templ\_string = 'Hey $name, there is a $error error!'

print(Template(templ\_string).substitute(name=name, error=hex(errno)))

# Вывод: 'Hey Bob, there is a 0xbadc0ffee error!'чную строку: