F-Строки: Новый улучшенный способ форматирования строк в Python

У нас для вас хорошие новости: f-строки вступают в дело, чтобы помочь с форматированием. Также известные как «**форматированные строковые литералы**», f-strings являются строковыми литералами с «**f**» в начале и фигурные скобки, содержащие выражения, которые в дальнейшем будут заменены своими значениями. Выражения оцениваются по мере выполнения и затем форматируются при помощи протокола **__format__** . Как всегда, документация Python может помочь, если хотите узнать больше.

Рассмотрим подробнее, как именно f-strings могут упростить вам жизнь.

Простой синтаксис

Синтаксис аналогичен тому, который вы используете в str.format(), но не такой перегруженный. Посмотрите на эту читабельность:

```python
name = "Eric"
age = 74

print(f"Hello, {name}. You are {age}.")
# Вывод: 'Hello, Eric. You are 74.'
```

Вы также можете использовать заглавную букву F:

```python
print(F"Hello, {name}. You are {age}.")
# Вывод: 'Hello, Eric. You are 74.'
```

Вам уже нравится? Надеемся, что да, в любом случае, вы будете в восторге к концу статьи.

Произвольные выражения

Так как f-строки оцениваются по мере выражения, вы можете внести любую или все доступные выражения Python в них. Это позволит вам делать интересные вещи, например следующее:

```python
print(f"{2 * 37}")
# Вывод: '74'
```

Также вы можете вызывать функции. Пример:

```python
def to_lowercase(input):
    return input.lower()

name = "Eric Idle"

print(f"{to_lowercase(name)} is funny.")
# Вывод: 'eric idle is funny.'
```

Также вы можете вызывать метод напрямую:

```python
print(f"{name.lower()} is funny.")
```

```
# Вывод: 'eric idle is funny.'
```

Вы даже можете использовать объекты, созданные из классов при помощи **f-строки**. Представим, что у вас есть следующий класс:

Python

```python
class Comedian:
    def __init__(self, first_name, last_name, age):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

    def __str__(self):
        return f"{self.first_name} {self.last_name} is {self.age}."

    def __repr__(self):
        return f"{self.first_name} {self.last_name} is {self.age}. Surprise!"
```

Вы могли бы сделать следующее:

Python

```python
new_comedian = Comedian("Eric", "Idle", "74")

print(f"{new_comedian}")
# Вывод: 'Eric Idle is 74.'
```

Методы **__str__**() и **__repr__**() работают с тем, как объекты отображаются в качестве строк, так что вам нужно убедиться в том, что вы используете один из этих методов в вашем **определении класса**. Если вы хотите выбрать один, попробуйте __repr__(), так как его можно использовать вместо __str__().

Строка, которая возвращается **__str__**() является неформальным строковым представлением объекта и должна быть читаемой. Строка, которую вернул __str__() — это официальное выражение и должно быть однозначным. При вызове **str**() и **repr**(), предпочтительнее использовать __str__() и __repr__() напрямую.

По умолчанию, **f-строки** будут использовать __str__(), но вы должны убедиться в том, что они используют __repr__(), если вы включаете флаг преобразования **!r**:

Python

```python
print(f"{new_comedian}")
# Вывод: 'Eric Idle is 74.'

print(f"{new_comedian!r}")
# Вывод: 'Eric Idle is 74. Surprise!'
```

Если вы хотите прочитать часть обсуждения, в результате которого **f-strings** поддерживают полные выражения Python, вы можете сделать это здесь.

Многострочные F-Strings

У вас могут быть многострочные f-strings:

```python
name = "Eric"
profession = "comedian"
affiliation = "Monty Python"

message = (
    f"Hi {name}. "
    f"You are a {profession}. "
    f"You were in {affiliation}."
)

print(message)
# Вывод: 'Hi Eric. You are a comedian. You were in Monty Python.'
```

Однако помните о том, что вам нужно разместить **f вначале каждой строки**. Следующий код **не будет** работать:

```python
message = (
    f"Hi {name}. "
    "You are a {profession}. "
    "You were in {affiliation}."
)

print(message)
# Вывод: 'Hi Eric. You are a {profession}. You were in {affiliation}.'
```

Если вы не внесете f в начале каждой индивидуальной строки, то получите обычную, старую версию строк, без приятных новшеств.

Если вы хотите размножить строки по нескольким линиям, у вас также есть возможность избежать возвратов при помощи \:

```python
message = f"Hi {name}. " \
          f"You are a {profession}. " \
          f"You were in {affiliation}."

print(message)
# Вывод: 'Hi Eric. You are a comedian. You were in Monty Python.'
```

Но вот что произойдет, если вы используете «»»:

```python
message = f"""
    Hi {name}.
    You are a {profession}.
```

```
    You were in {affiliation}.
"""

print(message)
# Вывод: '\n    Hi Eric.\n    You are a comedian.\n    You were in Monty Python.\n
```

Инструкция по отступам доступна в PEP 8.

Скорость

Буква f в **f-strings** может также означать и “**fast**”. Наши f-строки заметно быстрее чем % и str.format() форматирования. Как мы уже видели, **f-строки** являются выражениями, которые оцениваются по мере выполнения, а не постоянные значения. Вот выдержка из документации:

> “F-Строки предоставляют способ встраивания выражений внутри строковых литералов с минимальным синтаксисом. Стоит **обратить внимание** на то, что f-строка является выражением, которое оценивается по мере выполнения, а не постоянным значением. В исходном коде Python f-строки является литеральной строкой с префиксом f, которая содержит выражения внутри скобок. Выражения заменяются их значением.”

Во время выполнения, выражение внутри фигурных скобок оценивается в собственной области видимости Python и затем сопоставляется со строковой литеральной частью f-строки. После этого возвращается итоговая строка. В целом, это все.

Рассмотрим сравнение скорости:

Python
```
>>> import timeit
>>> timeit.timeit("""name = "Eric"
... age = 74
... '%s is %s.' % (name, age)""", number = 10000)

0.003324444866599663
```

Python
```
>>> timeit.timeit("""name = "Eric"
... age = 74
... '{} is {}.'.format(name, age)""", number = 10000)

0.004242089427570761
```

Python
```
>>> timeit.timeit("""name = "Eric"
... age = 74
... f'{name} is {age}.'""", number = 10000)

0.0024820892040722242
```

Как вы видите, f-строки являются самыми быстрыми.

Однако, суть не всегда в этом. После того, как они реализуются первыми, у них есть определенные проблемы со скоростью и их нужно сделать быстрее, чем **str.format**(). Для этого был предоставлен специальный опкод BUILD_STRING.

Python F-Строки: Детали

На данный момент мы узнали почему **f-строки** так хороши, так что вам уже может быть интересно их попробовать в работе. Рассмотрим несколько деталей, которые нужно учитывать:

Кавычки

Вы можете использовать несколько типов кавычек внутри выражений. Убедитесь в том, что вы не используете один и тот же тип кавычек внутри и снаружи **f-строки**.

Этот код будет работать:

```python
print(f"{'Eric Idle'}")
# Вывод: 'Eric Idle'
```

И этот тоже:

```python
print(f'{"Eric Idle"}')
# Вывод: 'Eric Idle'
```

Вы также можете использовать тройные кавычки:

```python
print(f"""Eric Idle""")
# Вывод: 'Eric Idle'
```

```python
print(f'''Eric Idle''')
# Вывод: 'Eric Idle'
```

Если вам понадобиться использовать один и тот же тип кавычек внутри и снаружи строки, вам может помочь \:

```python
print(f"The \"comedian\" is {name}, aged {age}.")
# Вывод: 'The "comedian" is Eric Idle, aged 74.'
```

Словари

Говоря о кавычках, будьте внимательны при работе со словарями Python. Вы можете вставить значение словаря по его ключу, но сам ключ нужно вставлять в одиночные кавычки внутри f-строки. Сама же f-строка должна иметь двойные кавычки.

Вот так:

```python
comedian = {'name': 'Eric Idle', 'age': 74}

print(f"The comedian is {comedian['name']}, aged {comedian['age']}.")
# Вывод: The comedian is Eric Idle, aged 74.
```

Обратите внимание на количество возможных проблем, если допустить ошибку в синтаксисе SyntaxError:

```python
>>> comedian = {'name': 'Eric Idle', 'age': 74}
>>> f'The comedian is {comedian['name']}, aged {comedian['age']}.'
  File "<stdin>", line 1
    f'The comedian is {comedian['name']}, aged {comedian['age']}.'
                                  ^
SyntaxError: invalid syntax
```

Если вы используете одиночные кавычки в ключах словаря и снаружи f-строк, тогда кавычка в начале ключа словаря будет интерпретирован как конец строки.

Скобки

Чтобы скобки появились в вашей строке, вам нужно использовать двойные скобки:

```python
print(f"{{74}}")

# Вывод: '{ 74 }'
```

Обратите внимание на то, что использование **тройных скобок** приведет к тому, что в строке будут только одинарные:

```python
print( f"{{{74}}}" )

# Вывод: '{ 74 }'
```

Однако, вы можете получить больше отображаемых скобок, если вы используете больше, чем три скобки:

```python
print(f"{{{{74}}}}")
```

```
# Вывод: '{{74}}'
```

Бэкслеши

Как вы видели ранее, вы можете использовать бэкслеши в части строки **f-string**. Однако, вы не можете использовать бэкслеши в части **выражения f-string**:

```Python
>>> f"{\"Eric Idle\"}"
  File "<stdin>", line 1
    f"{\"Eric Idle\"}"
                    ^
SyntaxError: f-string expression part cannot include a backslash
```

Вы можете проработать это, оценивая выражение заранее и используя результат в f-строк:

```Python
name = "Eric Idle"
print(f"{name}")

# Вывод: 'Eric Idle'
```

Междустрочные комментарии

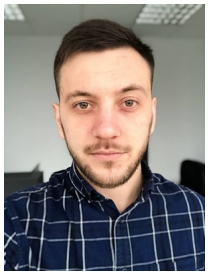
Выражения не должны включать комментарии с использованием **символа #**. В противном случае, у вас будет ошибка синтаксиса SyntaxError:

```Python
>>> f"Eric is {2 * 37 #Oh my!}."
  File "<stdin>", line 1
    f"Eric is {2 * 37 #Oh my!}."
                              ^
SyntaxError: f-string expression part cannot include '#'
```

Идите с миром и форматируйте!

Разумеется, вы можете использовать старые методы форматирования строк, но с **f-строками** у вас есть более лаконичный, читаемый и удобный способ, который одновременно и быстрее, и менее вероятно приведет к ошибке. Упростить свою жизнь используя **f-строки** — отлична причина пользоваться **Python 3.6**, если вы еще не перешли к этой версии. (Если вы все еще пользуетесь Python 2.7, не беспокойтесь, 2020 год не за горами!)

Согласно дзену Python, когда вам нужно выбрать способ решения задачи, всегда “есть один — и желательно только один очевидный способ сделать это”. Кстати, f-строки не являются единственным способом **форматирования строк**. Однако, их использование вполне может стать единственным адекватным способом.



Vasile Buldumac

Являюсь администратором нескольких порталов по обучению языков программирования Python, Golang и Kotlin. В составе небольшой команды единомышленников, мы занимаемся популяризацией языков программирования на русскоязычную аудиторию. Большая часть статей была адаптирована нами на русский язык и распространяется бесплатно.

E-mail: vasile.buldumac@ati.utm.md

Образование

Universitatea Tehnică a Moldovei (*utm.md*)

2014 — 2018 Технический Университет Молдовы, ИТ-Инженер. Тема дипломной работы «*Автоматизация покупки и продажи криптовалюты используя технический анализ*»

2018 — 2020 Технический Университет Молдовы, Магистр, Магистерская диссертация «*Идентификация человека в киберпространстве по фотографии лица*»