## MUNDARIJA

**Х.Ш. РУСТАМОВ**
Доцент кафедры
прикладная математика и технологий
программирования
Бухарского государственного университета

**М.А. БАБАДЖАНОВА**
Магистрант
прикладная математика
Бухарского государственного университета

# РАБОТА СО СТРОКОВЫМИ ВЕЛИЧИНАМИ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ PYTHON

*В этой статье представлен обзор строка в Python, а также примеры их использования. Было подробно рассмотрено множество различных механизмов, которые Python предоставляет для работы со строками, включая операторы, встроенные функции и встроенные методы.*

***Ключные слова:*** *“Старый стиль” форматирования, “Новый стиль” форматирования, f-Strings, Standard Library, Отступы и выравнивание строк, Datetime, Встроенные методы строк.*

*Ushbu maqolada Pythondagi satrlarning umumiy ko’rinishi va ulardan foydalanish misollari keltirilgan. Biz Python qatorlar bilan ishlash uchun taqdim etadigan turli xil mexanizmlarni, jumladan operatorlar, o’rnatilgan funksiyalar va o’rnatilgan usullarni batafsil ko’rib chiqdik.*

***Kalit so’zlar:*** *“Eski uslub” formatlash, “yangi uslub” formatlash, f-satrlar, standart kutubxona, satrlarni cheklash va tekislash, sana vaqti, o’rnatilgan qator usullari.*

*This article provides an overview of strings in Python, as well as examples of their use. The many different mechanisms that Python provides for working with strings were discussed in detail, including operators, built-in functions, and built-in methods.*

***Key words:*** *“Old style” formatting, “New style” formatting, f-Strings, Standard Library, Indenting and aligning strings, Datetime, Built-in string methods.*

Строки в языке Python являются типом данных, специально предназначенным для обработки текстовой информации. Строка может содержать произвольно длинный текст (ограниченный имеющейся памятью).

Существует четыре разных подхода к форматированию строк:

1. “Старый стиль” форматирования строк (оператор %)
2. “Новый стиль” форматирования строк (str.format)
3. Интерполяция строк / f-Strings (Python 3.6+)
4. Шаблоны (Standard Library)

**“Старый стиль” форматирования строк (оператор %)**

```
name = “Eric”
“Hello, %s." % name
```

Результат: Hello, Eric.

Поскольку оператор % принимает только один аргумент, для случая с несколькими переменными нужно обернуть правую часть в кортеж, например, так:

```
name = “Eric”
age = 74
“Hello, %s. You are %s.” % (name, age)
```

Hello, Eric. You are 74.

Если переменных много, что код быстро становиться плохо читаемым:

```
first_name = “Eric”
last_name = “Idle”
age = 74
profession = “comedian”
affiliation = “Monty Python”
“Hello, %s %s. You are %s. You are a %s. You were a member of %s.” % (first_name, last_name, age, profession, affiliation)
```

Hello, Eric Idle. You are 74. You are a comedian. You were a member of Monty Python.

Также можно производить подстановку переменных по имени:

```
‘Hey %(name)s, there is a 0x%(errno)x error!’ % {“name”: name, “errno”: errno }
```

Hey Bob, there is a 0xbadc0ffee error!

К сожалению, этот вид форматирования не очень хорош, потому что он многословен и приводит к ошибкам, таким как неправильное отображение кортежей или словарей.

**“Новый стиль” (str.format)**

str.format () - это улучшение % форматирования. Он использует обычный синтаксис вызова функции и может быть расширен с помощью переопределения метода __format__() для объекта, преобразуемого в строку.

https://www.python.org/dev/peps/pep-3101/#controlling-formatting-on-a-per-type-basis

Был представлен в Python 2.6

https://docs.python.org/3/library/stdtypes.html#str.format

Поля замены отмечены фигурными скобками:

“Hello, {}. You are {}.”.format(name, age)

Hello, Bob. You are 74.

Можно ссылаться на переменные в любом порядке, используя их индекс:

“Hello, {1}. You are {0}.”.format(age, name)

Hello, Bob. You are 74.

Но если вы вставите имена переменных, вы получите дополнительную возможность передавать объекты, а затем ссылаться на параметры и методы между фигурными скобками:

person = {‘name’: ‘Eric’, ‘age’: 74}

“Hello, {name}. You are {age}.”.format(name=person[‘name’], age=person[‘age’])

Hello, Eric. You are 74.

Также часто удобно использовать ** , для вывода значений из словаря:

“Hello, {name}. You are {age}.”.format(**person)

Hello, Eric. You are 74.

Недостатки: Хотя код, использующий str.format (), гораздо легче читается, в сравнении с %-форматированием, однако str.format () все еще может быть слишком громоздким, когда строка длинная и параметров много.

```
first_name = “Eric”
last_name = “Idle”
age = 74
profession = “comedian”
affiliation = “Monty Python”
print((“Hello, {first_name} {last_name}. You are {age}. ” +
“You are a {profession}. You were a member of {affiliation}.”) \
.format(first_name=first_name, last_name=last_name, age=age, \
profession=profession, affiliation=affiliation))
```

Hello, Eric Idle. You are 74. You are a comedian. You were a member of Monty Python.

**f-Strings / Интерполяция строк**

f-Strings -новый и улучшенный способ форматирования строк в Python.

Поддержка добавлена начиная с Python 3.6. Вы можете прочитать все об этом в PEP 498. https://www.python.org/dev/peps/pep-0498/

```
name = “Eric”
age = 74
f”Hello, {name}. You are {age}.”
```

Hello, Eric. You are 74.

Поскольку f-строки вычисляются во время выполнения, в них можно поместить все допустимые выражения Python.

```
print( f”{2 * 37}” )
```

74

```
def to_lowercase(input):
 return input.lower()
name = “Eric Idle”
print(f”{to_lowercase(name)} is funny.”)
```

eric idle is funny.

Можно вызвать метод напрямую:

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

eric idle is funny.

Другой пример:

```
print(f’Если поделить торт на трех человек, каждый из них получит {1/3*100:.2f} %’)
```

Если поделить торт на трех человек, каждый из них получит 33.33 %

Пример с классом:

```
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!"
new_comedian = Comedian("Eric", "Idle", "74")
print(f"{new_comedian}")
Eric Idle is 74.
print(f"{new_comedian!r}")
Eric Idle is 74. Surprise!
```

Многострочные f-строки

```
name = "Eric"
profession = "comedian"
affiliation = "Monty Python"
message = (
 f "Hi {name}."
 f"You are a {profession}."
 F"You were in {affiliation}."
 )
print(message)
Hi Eric. You are a comedian. You were in Monty Python.
```

Важно, чтобы f было перед каждой строкой многострочной строки. Следующий код не будет работать:

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

При использовании """ строк:

```
message = f """
 Hi {name}.
 You are a {profession}.
 You were in {affiliation}.
"""
print(message)
Hi Eric.
You are a comedian.
You were in Monty Python.
```

**Скорость**

Символ f в f-string также может означать "быстро" (fast).

f-строки быстрее, чем %-форматирование и str.format().

f-строки - это выражения, вычисляемые во время выполнения.

```
import timeit
timeit.timeit("""name = "Eric" age = 74 '%s is %s.' % (name, age) """, number = 10000)
0.004173202378340368
timeit.timeit("""name = "Eric" age = 74 '{} is {}.'.format(name, age) """, number = 10000)
0.00488798551082123
timeit.timeit("""name = "Eric" age = 74 f'{name} is {age}.'""", number = 10000)
0.0025852118251977196
```

Может быть потенциально небезопасно, если строка получается от пользователя.

```
# Это наш супер секретный ключ:
SECRET = 'this-is-a-secret'
class Error:
 def __init__(self):
 pass
# Злонамеренный пользователь может ввести строку, которая
# может прочитать данные из глобального пространства имен:
user_input = '{error.__init__.__globals__[SECRET]}'
# Что позволит получить секретную информацию,
# такую как секретный ключ:
err = Error()
print(user_input.format(error=err))
this-is-a-secret
```

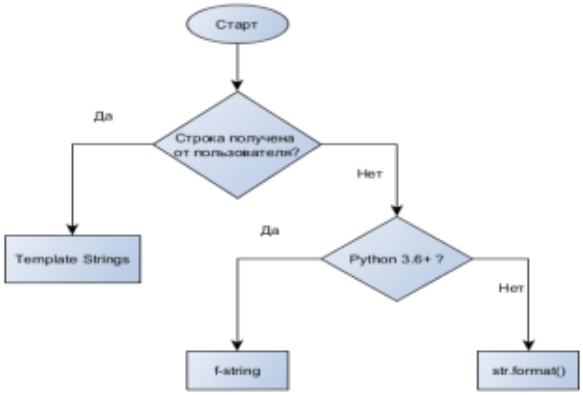
**Шаблонные строки (Стандартная библиотека Template Strings)**

Еще один инструмент для форматирования строк в Python: шаблоны. Был добавлен в Python 2.4. Это более простой и менее мощный механизм, но в некоторых случаях это может быть именно то, что нужно.

```
from string import Template
t = Template('Hey, $name!')
s = t.substitute(name=name)
print(s)
Hey, Eric Idle!
```

**Каким методом форматирования строк стоит пользоваться?**

Если строки получены от пользователя, используйте шаблонные строки (способ 4), чтобы избежать проблем с уязвимостью программы.




Другое отличие состоит в том, что строки шаблона не допускают спецификаторов формата.

```
templ_string = 'Hey $name, there is a $error error!'
s = Template(templ_string).substitute(name=name, error=hex(errno))
print(s)
Hey Eric Idle, there is a 0xbadc0ffee error!
```

Добавить отступы слева:

```
test = 'test'
print('%10s' % (test,))
print('{:>10}'.format(test))
print(f'{test:>10}')
 test
 test
 test
```

Добавить отступы справа:

```
test = 'test'
print('{:_<10}'.format(test))
print(f'{test:_<10}')
test______
test______
```

Выравнивание по центру:

Если при выравнивании по центру получается нечетное количество отступов, то нечетный будет добавлен справа.

```
test = 'test'
print('{:_^10}'.format(test))
print(f'{test:_^10}')
___test___
___test___
print('{:_^6}'.format('zip'))
_zip__
Целые числа:
print('%d' % (42,))
print('{:d}'.format(42))
print(f'{42:d}')
42
42
42
```

```
Floats:
print('%f' % (3.141592653589793,))
print('{:f}'.format(3.141592653589793))
print(f'{3.141592653589793:f}')
3.141593
3.141593
3.141593
pi = 3.141592653589793
print('%06.2f' % (pi))
print('{:06.2f}'.format(pi))
print(f'{pi:06.2f}')
003.14
003.14
003.14
```

Пробел означает, что для отрицательного значения будет отображен минус, а для положительного пробел.

```
print('{: d}'.format((- 23)))
print('{: d}'.format((23)))
-23
 23
print(f'{-23: d}')
print(f'{23: d}')
-23
 23
```

**Datetime**

```
from datetime import datetime
s = '{:%Y-%m-%d %H:%M}'.format(datetime(2001, 2, 3, 4, 5))
print(s)
2001-02-03 04:05
Datetime (f-строки):
from datetime import datetime
dt = datetime(2001, 2, 3, 4, 5)
s = f'{dt:%Y-%m-%d %H:%M}'
print(s)
2001-02-03 04:05
```

**Встроенные методы строк в python**

• string.capitalize()-приводит первую букву в верхний регистр, остальные в нижний.

Возвращает копию s с первым символом, преобразованным в верхний регистр, и остальными символами, преобразованными в нижний регистр:

```
s = 'everyTHing yoU Can IMaGine is rEAl'
s.capitalize()
'Everything you can imagine is real'
#   Не алфавитные символы не изменяются:
s = 'follow us @PYTHON'
s.capitalize()
'Follow us @python'
```

• string.lower()-преобразует все буквенные символы в строчные.

Возвращает копию s со всеми буквенными символами, преобразованными в нижний регистр:

```
s = 'everyTHing yoU Can IMaGine is rEAl'
s.lower()
'everything you can imagine is real'
```

• string.swapcase()-возвращает копию s с заглавными буквенными символами, преобразованными в строчные и наоборот:

```
s = 'everyTHing yoU Can IMaGine is rEAl'
```

s.swapcase()
'EVERYthING YOu cAN imAgINE IS ReaL'

• string.title()-преобразует первые буквы всех слов в заглавные.

возвращает копию, s в которой первая буква каждого слова преобразуется в верхний регистр, а остальные буквы — в нижний регистр:

s = 'the sun also rises'
s.title()
'The Sun Also Rises'

• string.upper()-преобразует все буквенные символы в заглавные.

Возвращает копию s со всеми буквенными символами в верхнем регистре:

s = 'follow us @PYTHON'
s.upper()
'FOLLOW US @PYTHON'

• string.count(<sub>[, <start>[, <end>]])-подсчитывает количество вхождений подстроки в строку.

Возвращает количество точных вхождений подстроки <sub> в s:

'foo goo moo'.count('oo')
3
# Количество вхождений изменится, если указать <start> и <end>:
'foo goo moo'.count('oo', 0, 8)
2

• string.endswith(<suffix>[, <start>[, <end>]])-определяет, заканчивается ли строка заданной подстрокой.

Возвращает, True если s заканчивается указанным <suffix> и False если нет:

>>> 'python'.endswith('on')
True
>>> 'python'.endswith('or')
False
# Сравнение ограничено подстрокой, между <start> и <end>, если они указаны:
>>> 'python'.endswith('yt', 0, 4)
True
>>> 'python'.endswith('yt', 2, 4)
False

Было подробно рассмотрено множество различных механизмов, которые Python предоставляет для работы со строками, включая операторы, встроенные функции и встроенные методы.

Рассмотренных стандартных возможностей для работы с текстом достаточно далеко не всегда. Например, в методах find() и replace() задается всего одна строка. В реальных задачах такая однозначность встречается довольно редко, чаще требуется найти или заменить строки, отвечающие некоторому шаблону.

Такую возможность предоставляют регулярные выражения, работа с которыми будет рассмотрена далее.

## Литературы


1. Федоров Д.Ю. Основы Программирования На примере языка Python // Наука и техника. - Санкт-Петербург, 2019. 25-32 стр.
2. Gabor Szabo. 1000 Python Examples. 2020, 52-66 pages.
3. https://metanit.com/python
4. https://pythonworld.ru/tipy-dannyx-v-python/stroki-funkcii-i-metody-strok.html
5. https://pythonru.com/osnovy/stroki-python