IBM

developerWorks Россия Технические материалы Open source Статьи

Программирование на Python. Часть 2: Строки в питоне

Предыдущая статья была посвящена основным возможностям Python. Сейчас мы рассмотрим один из базовых типов этого языка программирования – строковый тип.

02.08.2010

Строкам в питоне присуща простота использования, наличие большого количества встроенных методов, разнообразие возможностей для гибкого использования в повседневной работе. Будут рассмотрены следующие аспекты.

1. Строковый тип.
2. Срезы (slicing).
3. Операции со строками.
4. Unicode.
5. Форматирование.
6. Встроенные методы.
7. Тест на конкатенацию.

1. Строковый тип

Строка – это последовательность символов с произвольным доступом. Строки в языке Python невозможно изменить – в этом случае говорят, что это immutable тип. Попытка изменить символ в определенной позиции или подстроку вызовет ошибку:

```
>>> word = 'strength'
>>> word[2] = 'y'
TypeError: 'str' object does not support item assignment
```

Но если очень хочется, то изменить можно, например, так:

```
>>> word = word[:3] + '!' + word[4:]
'str!ngth'
```

Или так:

```
>>> word = word.replace('!','e')
'strength'
```

Индексы могут иметь отрицательные значения для отсчета с конца – отсчет начинается с -1:

```
>>> word[-1]
h
```

Строки в питоне можно заключать как в одинарные, так и в двойные кавычки, причем кавычки одного типа могут быть произвольно вложены в кавычки другого типа:

```
>>> '123'
'123'
>>> "7'8''9"
"7'8''9"
```

Длинные строки можно разбивать на несколько строк с помощью обратного слеша:

```
>>> s = 'this is first word\
and this is second word'
```

Большие наборы строк и целые тексты можно заключать в тройные кавычки:

```
>>> print """
One
Two
Three
"""
```

Обратный слеш в строках используется для так называемой escape-последовательности.

После слеша может идти один или несколько символов.

В следующем примере комбинация '\n' – это новая строка, '\t' – это табуляция:

```
>>> s ='a\nb\tc'
>>> print s
a
b       c
```

В следующем примере строка состоит из бинарной последовательности трех чисел – двух восьмеричных и одного шестнадцатеричного:

```
>>> s = '\001\002\x03'
>>> s
'\x01\x02\x03'
>>> len(s)
3
```

Нужно заметить, что питоновские строки вообще не терминируются в конце нулевым байтом, как это делается, например, в си.

2. Срезы

Срез – это механизм гибкого управления строкой на основе индексации. Можно получить любой символ строки по его индексу. Подобно си, первый символ имеет индекс 0. Подстрока может быть определена с помощью среза – двух индексов, разделенных двоеточием:

```
>>> word = 'strength'
>>> word[4]
n
>>> word[0:2]
st
>>> word[2:4]
re
```

Если в срезе опущен первый символ, значит, он равен нулю; если опущен последний символ – он равен длине строки:

```
>>> word[:3]
str
>>> word[5:]
gth
```

Можно выбирать последовательность символов из строки с определенной цикличностью:

```
>>> s = '1234567890'
>>> s[::2]
'13579'
>>> s[1:10:2]
'13579'
>>> s[::-1]
'0987654321'
```

3. Операции со строками

Строки можно склеивать с помощью оператора + :

```
>>> var = 'Moscow' + 'city'
```

Между двух строк подряд вообще можно ничего не ставить, и они будут сконкатенированы.

Строки можно умножать с помощью оператора * :

```
>>> '123' * 3
'123123123'
```

Строки можно сравнивать с помощью операторов <, <=, ==, !=, >, >=.

4. Unicode

Unicode позволяет применять все символы, используемые в текстах на разных языках. Ранее мы могли использовать только 256 символов из определенной кодовой страницы. Перед строкой нужно поставить спецификатор u – при этом на каждый символ отводится 2 байта, так как по умолчанию ставится кодировка UTF-8:

```
>>> w = u'Вася Пупкин'
>>> w
u'\u0412\u0430\u0441\u044f \u041f\u0443\u043f\u043a\u0438\u043d'
```

Юникодную строку можно также создать в известной кодировке: ASCII, UTF-8, UTF-16, KOI8-R, CP1251, CP866 и т.д.:

```
>>> s = unicode("Привет", "KOI8-R")
>>> s
u'\u043f\xf7\u044f\u2500\u043f\u2566\u043f\u2561\u043f\u2563\u044f\u250c'
```

Метод encode() позволяет преобразовывать строки Unicode в обычные строки, содержащие текст в указанной кодировке:

```
>>> s.encode("KOI8-R")
'\xd0\x9f\xd1\x80\xd0\xb8\xd0\xb2\xd0\xb5\xd1\x82'
```

5. Форматирование

Форматирование в питоне – мощный инструмент управления строками. Есть несколько подходов – стандартный и с использованием шаблонов. Для форматирования в питоновских строках используется стандартный оператор – символ %. Слева от процента указываем строку, справа – значение или список значений:

```
>>> s = 'Hello %s' % 'word'
>>> s
'Hello word'

>>> s = 'one  %s %s' % ('two','three')
>>> s
'one  two three'
```

Если нужно преобразование числа в строку, используется числовой спецификатор – %d или %f:

```
>>> s = 'one  %d %f' % (2 , 3.5)
>>> s
'one  2 3.500000'
```

При форматировании можно указать общую ширину строки и точность для чисел, при этом число будет дополнено незначащими нулями. В следующем примере результирующая строка будет иметь длину 10 символов, на дробную часть будет отведено 5 символов:

```
>>> x = 4/3
>>> '%10.5f' % x
'   1.00000'
```

Пробелы слева можно отформатировать нулями:

```
>>> from math import pi
>>> '%015.10f' % pi
'0003.1415926536'
```

Для форматирования можно использовать другой подход – шаблоны строк, Template. В следующем примере для форматирования уже можно использовать словарь:

```
>>> from string import Template
>>> s = Template('1 $two 3 4 $five')
>>> d={}
>>> d['two']=2
>>> d['five']=5
>>> s.substitute(d)
'1 2 3 4 5'
```

Таблица типов форматирования для строк

Код	Значение
s	Строковый

Код	Значение
r	Строковый, но с использованием `repr`, а не `str`
c	Посимвольный
d	Десятичный
i	Целый
u	То же, что и d (`no longer unsigned`)
o	Восьмеричный
x	Шестнадцатеричный
X	Шестнадцатеричный в верхнем регистре
e	`Floating-point exponent`, нижний регистр
E	То же, что и e, но в верхнем регистре
f	`Floating-point decimal`
F	`Floating-point decimal`
g	`Floating-point e` или `f`
C	`Floating-point E` или `F`
%	Символьный %

6. Методы

Строки обладают большим набором разнообразных методов. Наиболее популярные из них:

`find` – находит подстроку в строке – возвращает позицию вхождения строки, либо `-1`:

```
>>> s = 'The find method finds a substring'
>>> s.find('find')
4
>>> s.find('finds')
16
>>> s.find('findsa')
-1
```

`join` – объединяет через разделитель набор строк:

```
>>> seq = ['one','two','three']
>>> sep = ','
>>> sep.join(seq)
'one,two,three'
```

`split` – это обратная функция для `join`, разбивает строку на последовательность:

```
>>> s = '/usr/local/bin'
>>> s.split('/')
['', 'usr', 'local', 'bin']
```

`replace` – заменяет в строке одну подстроку на другую:

```
>>> s = 'replace method returns a string'
>>> s.replace('returns','return')
'replace method return a string'
```

`strip` – удаляет пробелы слева и справа:

```
>>> '       this is whitespace string    '.strip()
'this is whitespace string'
```

`translate` – в отличие от `replace`, может делать множественную замену. В следующем примере каждый символ '1' в исходной строке будет заменен на символ '3', а символ '2' – на символ '4' соответственно:

```
>>> from string import maketrans
>>> table = maketrans('12', '34')
>>> '1212 5656'.translate(table)
'3434 5656'
```

Для конверсии различных типов в строковый используются функции `str, int, ord, chr`:

`str` – конвертирует число в строку;

`int` – конвертирует строку в число;

`ord` – возвращает значение байта;

`chr` – конвертирует число в символ.

Таблица методов, доступных в Python 3.0

`S.capitalize()` `S.ljust(width [, fill])`
`S.center(width [, fill])` `S.lower()`
`S.count(sub [, start [, end]])` `S.lstrip([chars])`
`S.encode([encoding [,errors]])` `S.maketrans(x[, y[, z]])`
`S.endswith(suffix [, start [, end]])` `S.partition(sep)`
`S.expandtabs([tabsize])` `S.replace(old, new [, count])`
`S.find(sub [, start [, end]])` `S.rfind(sub [,start [,end]])`
`S.format(fmtstr, *args, **kwargs)` `S.rindex(sub [, start [, end]])`
`S.index(sub [, start [, end]])` `S.rjust(width [, fill])`
`S.isalnum()` `S.rpartition(sep)`
`S.isalpha()` `S.rsplit([sep[, maxsplit]])`
`S.isdecimal()` `S.rstrip([chars])`
`S.isdigit()` `S.split([sep [,maxsplit]])`
`S.isidentifier()` `S.splitlines([keepends])`
`S.islower()` `S.startswith(prefix [, start [, end]])`
`S.isnumeric()` `S.strip([chars])`
`S.isprintable()` `S.swapcase()`
`S.isspace()` `S.title()`

`S.capitalize()` `S.ljust(width [, fill])`
`S.istitle()` `S.translate(map)`
`S.isupper()` `S.upper()`
`S.join(iterable)` `S.zfill(width)`

7. Тест на конкатенацию

Тест на строковую конкатенацию в питоне дает интересные результаты. Стандартный подход, при котором новый сегмент добавляется к концу уже существующей строки, в питоне неэффективен: питон при каждой конкатенации будет создавать новый объект, что очень медленно.

Проведем несложный тест: к строке будем конкатенировать символьное представление натуральных чисел по возрастанию в диапазоне от 0 до миллиона.

1-й метод. Используем для хранения строки массив символов `array`. Будем просто добавлять туда строковое представление натурального числа, а в конце применим метод `array.tostring()`. Этот метод оказался самым медленным.

2-й метод. Используем тот же алгоритм, что и в первом методе, только вместо массива array используем список, а в конце сделаем стандартный джойн.

3-й метод. Используем модуль `cStringIO`, в котором есть возможность писать в псевдо-файл, который на самом деле хранится в памяти. В конце вызываем метод `getvalue()`.

4-й метод. Создаем в цикле список символьных представлений чисел, а потом одним махом джойним этот список. Отличие его от второго метода в том, что не используется `append()`. Вы увидите, что этот метод – самый быстрый.

Код:

```
import time

loop_count = 1000000

def method1():
  from array import array
  char_array = array('c')
  for num in xrange(loop_count):
    char_array.fromstring(`num`)
  return char_array.tostring()

def method2():
  str_list = []
  for num in xrange(loop_count):
    str_list.append(`num`)
  return ''.join(str_list)

def method3():
  from cStringIO import StringIO
  file_str = StringIO()
  for num in xrange(loop_count):
    file_str.write(`num`)
  return file_str.getvalue()

def method4():
  return ''.join([`num` for num in xrange(loop_count)])

t1 = time.time()
method1()
t2 = time.time()
print "\t%.1f" % ((t2 - t1))
method2()
t3 = time.time()
print "\t%.1f" % ((t3 - t2))
method3()
t4 = time.time()
print "\t%.1f" % ((t4 - t3))
method4()
t5 = time.time()
print "\t%.1f" % ((t5 - t4))
```

Подведение итогов

Строки относятся к наиболее популярным базовым типам. Срезы, большой набор встроенных функций, удобное форматирование позволяет гибко и оперативно производить манипуляции там, где мы могли бы затратить значительно большее количество времени на рутинные операции, будь это какой-то другой язык. Питон — это удобство, простота, минимальное количество усилий. В продолжение цикла речь пойдет о списках и словарях. Затем поговорим о модулях, классах и работе с файловой системой средствами Python. Код примеров проверялся на версии питона 2.6.