Строки

Глава 6

Python for Informatics: Exploring Information
www.pythonlearn.com

Строковый тип данных

- Строка представляет собой последовательность символов
- Строковый литерал использует кавычки 'Hello' или "Hello"
- Оператор “+” выполняет “конкатенацию” строк
- Если в строке содержится число, это все равно строка
- С помощью функции int() можно преобразовать строку с числом в число

```
>>> str1 = "Hello"
>>> str2 = 'there'
>>> bob = str1 + str2
>>> print bob
Hellothere
>>> str3 = '123'
>>> str3 = str3 + 1
Traceback (most recent call last):  File "<stdin>", line 1, in <module>TypeError: cannot concatenate 'str' and 'int' objects
>>> x = int(str3) + 1
>>> print x
124
>>>
```

Чтение и преобразование

- Мы предпочитаем читать данные как строки, а затем разбирать и преобразовывать данные по мере необходимости
- Это помогает контролировать ошибки и/или неверно введенные пользователем данные
- Введенные пользователем числа необходимо преобразовать из строкового типа в числовой

```
>>> name = raw_input('Введите:')
Введите:Chuck
>>> print name
Chuck
>>> apple = raw_input('Введите:')
Введите:100
>>> x = apple - 10
Traceback (most recent call
last):  File "<stdin>", line 1,
in <module>TypeError: unsupported
operand type(s) for -: 'str' and
'int'
>>> x = int(apple) - 10
>>> print x
90
```

Разбор строк

- Мы можем получить любой символ строки по его индексу, который указывается в квадратных скобках
- Значение индекса должно быть целым, а отсчет индексов начинается с нуля
- Значение индекса можно представить в виде математического выражения

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruit = 'banana'
>>> letter = fruit[1]
>>> print letter
a
>>> n = 3
>>> w = fruit[n - 1]
>>> print w
n
```

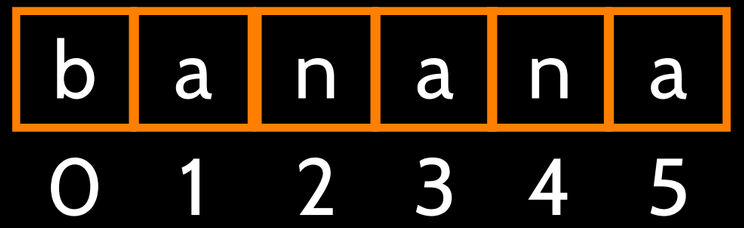
На символ дальше

- При указании индекса, превышающего длину строки, Python выдаст ошибку
- Будьте внимательны при вводе значений индексов и срезов

```
>>> zot = 'abc'
>>> print zot[5]
Traceback (most recent call last):  File "<stdin>", line 1, in <module>IndexError: string index out of range
>>>
```

Строки имеют длину

- Существует встроенная функция len, которая возвращает число символов в строке




```
>>> fruit = 'banana'
>>> print len(fruit)
6
```

Функция len

```
>>> fruit = 'banana'
>>> x = len(fruit)
>>> print x
6
```

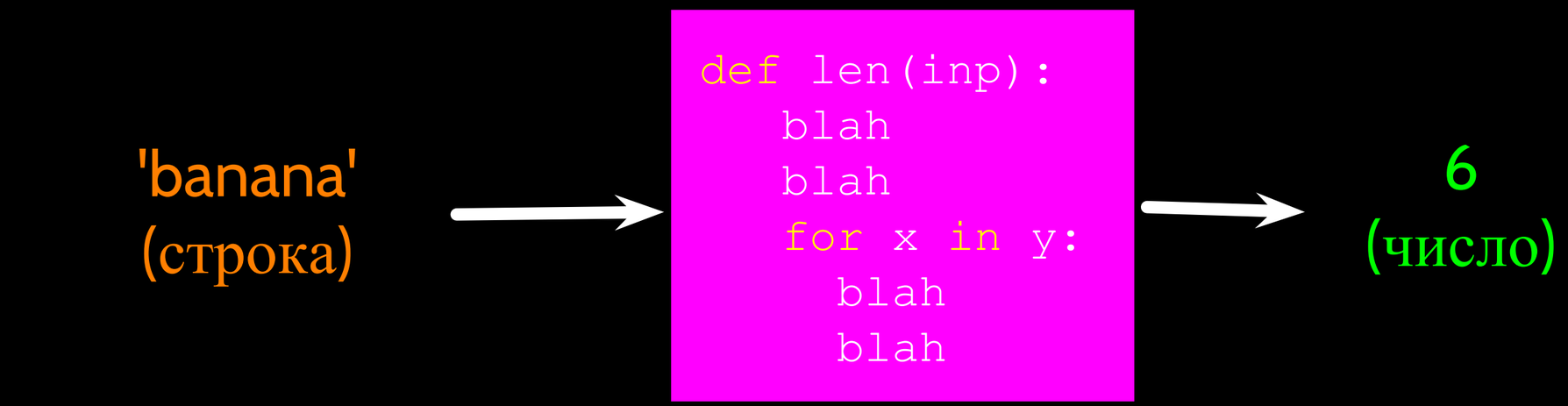
Функция - это встроенный код. Функция принимает некоторые данные на входе и производит результат.

Этот код написал г-н Гвидо ван Россум

Функция len

```
>>> fruit = 'banana'
>>> x = len(fruit)
>>> print x
6
```

Функция - это встроенный код. Функция принимает некоторые данные на входе и производит результат.



Циклы со строками

- С помощью инструкции while, итерационной переменной и функции len можно создать цикл, который по отдельности проходит через каждый символ строки

```
fruit = 'banana'
index = 0
while index < len(fruit):
    letter = fruit[index]
     print index, letter
     index = index + 1
```

```
0 b
1 a
2 n
3 a
4 n
5 a
```

Циклы со строками

- Определенный цикл с инструкцией for подходит гораздо лучше
- Цикл с инструкцией for отменяет необходимость использования итерационной переменной

```
fruit = 'banana'
for letter in fruit:
    print letter
```

```
b
a
n
a
n
a
```

Циклы со строками

- Определенный цикл с инструкцией for подходит гораздо лучше

- Цикл с инструкцией for отменяет необходимость использования итерационной переменной

```
fruit = 'banana'
for letter in fruit :
    print letter
```

```
index = 0
while index < len(fruit) :
    letter = fruit[index]
    print letter
    index = index + 1
```

```
b
a
n
a
n
a
```

Циклы и подсчет

- Этот простой цикл проходит через каждую букву в строке и подсчитывает, сколько раз встречается буква “a”

```
word = 'banana'
count = 0
for letter in word :
    if letter == 'a' :
        count = count + 1
print count
```

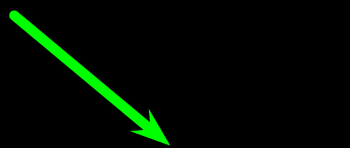
Оператор in

- Итерационная переменная поочередно проходит по последовательности (упорядоченному набору) данных
- Блок (тело) цикла выполняется один раз для каждого значения в (in) последовательности

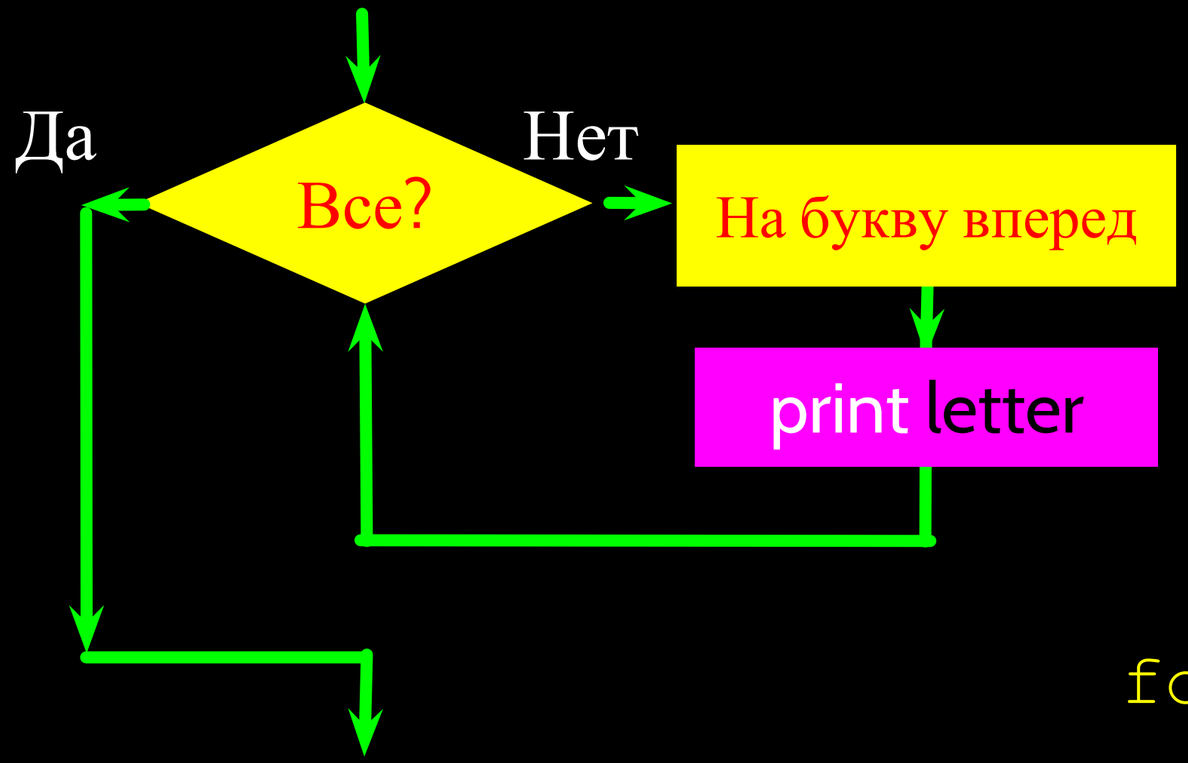
- Итерационная переменная поочередно указывает на каждый элемент в последовательности

Итерационная переменная

Строка из 6 символов



```
for letter in 'banana' :
    print letter
```



b	a	n	a	n	a

```
for letter in 'banana' :
    print letter
```

Итерационная переменная проходит через всю строку, и блок (тело) цикла выполняется один раз для каждого значения в последовательности

- С помощью двоеточия мы можем отделить любую непрерывную часть строки
- Второе число в срез не включается, то есть срезать "до, но не включая"
- Если второе число выходит за конец строки, срез останавливается в конце строки

```
>>> s = 'Monty Python'
>>> print s[0:4]
Mont
>>> print s[6:7]
P
>>> print s[6:20]
Python
```

Срез строк

- Опущенный первый или последний индекс означает начало или конец строки соответственно

```
>>> s = 'Monty Python'
>>> print s[:2]
Mo
>>> print s[8:]
Thon
>>> print s[:]
Monty Python
```

Срез строк

Объединение строк

- Знаком **+** в строках обозначается объединение

```
>>> a = 'Hello'
>>> b = a + 'There'
>>> print b
HelloThere
>>> c = a + '  ' + 'There'
>>> print c
Hello There
>>>
```

in как оператор

- Ключевое слово in также используется для проверки содержания одной строки “в” другой
- Выражение с in является логическим и выдает результат True (истинно) или False (ложно). Выражение с in можно использовать в инструкции if

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...      print 'Found it!'
...
Found it!
>>>
```

Сравнение строк

```
if word == 'banana':
    print  'All right, bananas.'

if word < 'banana':
    print 'Your word,' + word + ', comes before banana.'
elif word > 'banana':
    print 'Your word,' + word + ', comes after banana.'
else:
    print 'All right, bananas.'
```

Библиотека строк

- В библиотеке Python содержится набор строковых функций
- Эти функции уже *встроены* в каждую строку, то есть мы используем их, добавляя необходимую функцию к строковой переменной
- Эти функции не изменяют исходную строку, а возвращают новую измененную строку

```
>>> greet = 'Hello Bob'
>>> zap = greet.lower()
>>> print zap
hello bob
>>> print greet
Hello Bob>
>> print 'Hi There'.lower()
hi there
>>>
```

```
>>> stuff = 'Hello world'
>>> type(stuff)<type 'str'>
>>> dir(stuff)
['capitalize', 'center', 'count', 'decode', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'index',
'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'partition', 'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
```

`str.`**`replace`**(*old*, *new*[, *count*])

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

`str.`**`rfind`**(*sub*[, *start*[, *end*]])

Return the highest index in the string where substring *sub* is found, such that *sub* is contained within s[start,end]. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` on failure.

`str.`**`rindex`**(*sub*[, *start*[, *end*]])

Like `rfind()` but raises `ValueError` when the substring *sub* is not found.

`str.`**`rjust`**(*width*[, *fillchar*])

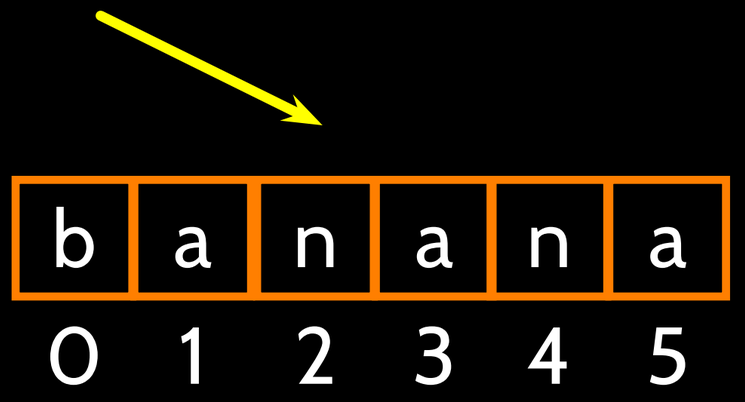
Return the string right justified in a string of length *width*. Padding is done using the specified *fillchar* (default is a space). The original string is returned if *width* is less than `len(s)`.

Библиотека строк

str.capitalize()
str.center(width[, fillchar])
str.endswith(suffix[, start[, end]])
str.find(sub[, start[, end]])
str.lstrip([chars])
str.replace(old, new[, count])
str.lower()
str.rstrip([chars])
str.strip([chars])
str.upper()

Поиск строки

- Функция find() используется для поиска подстроки в строке
- find() находит первое вхождение указанной подстроки
- Если подстрока не найдена, find() выдает -1
- Не забудьте, что отсчет элементов строки начинается с нуля




```
>>> fruit = 'banana'
>>> pos = fruit.find('na')
>>> print pos
2
>>> aa = fruit.find('z')
>>> print aa
-1
```

Преобразование в ВЕРХНИЙ РЕГИСТР

- Можно преобразовать строку в нижний или верхний регистр
- Часто выполняется поиск строки при помощи функции find(). Для этого мы сначала преобразуем строку в нижний регистр и таким образом выполняем поиск вне зависимости от регистра

```
>>> greet = 'Hello Bob'
>>> nnn = greet.upper()
>>> print nnn
HELLO BOB
>>> www = greet.lower()
>>> print www
hello bob
>>>
```

Поиск и замена

- Функция replace() похожа на оператор “поиска и замены” в текстовом редакторе
- Она заменяет все случаи искомой строки на строку замены

```
>>> greet = 'Hello Bob'
>>> nstr = greet.replace('Bob','Jane')
>>> print nstr
Hello Jane
>>> nstr = greet.replace('o','X')
>>> print nstr
HellX BXb
>>>
```

Удаление пробелов

- Иногда нам необходимо удалить лишние пробелы в начале и(или) конце строки
- lstrip() и rstrip() удаляют пробелы слева и справа соответственно
- strip() удаляет пробелы и в начале, и в конце строки

```
>>> greet = '   Hello Bob  '
>>> greet.lstrip()
'Hello Bob  '
>>> greet.rstrip()
'   Hello Bob'
>>> greet.strip()
'Hello Bob'
>>>
```

Префиксы

```
>>> line = 'Please have a nice day'
>>> line.startswith('Please')
True
>>> line.startswith('p')
False
```

Разбор и извлечение

21 31

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16
2008'
>>> atpos = data.find('@')
>>> print atpos
21
>>> sppos = data.find(' ',atpos)
>>> print sppos
31
>>> host = data[atpos+1 : sppos]
>>> print host
uct.ac.za
```

Обзор

- Строковый тип
- Чтение/преобразование
- Индексация строк []
- Срезы строк [2:4]
- Выполнение циклов по строкам с помощью for и while
- Объединение строк при помощи знака +
- Операции со строками
- Библиотека строк
- Сравнение строк
- Поиск в строках
- Замена текста
- Удаление пробелов

Благодарность / Содействие

Первоначальная разработка: Чарльз Северанс, Школа информации Мичиганского университета

Здесь впишите дополнительных авторов и переводчиков...

...