Строки

Глава 6

Пайтон для всех
www.py4e.com

Строковый тип данных

- Строка — это последовательность символов алфавита
- Строка заключается в одинарные или двойные кавычки: 'Привет' или "Привет"
- В случае строк, символ + означает «объединение»
- Если в строке содержатся цифры, она все равно остается строкой
- Мы можем преобразовать цифры в строке в числовой тип, используя функцию int()

```
>>> str1 = "Привет"
>>> str2 = 'тебе'
>>> bob = str1 + str2
>>> print(bob)
Приветтебе
>>> 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, используя строки, а затем анализировать и преобразовывать данные по мере необходимости
- Так мы имеем больше контроля над ошибками и/или неправильным вводом данных пользователем
- Введенные числа должны быть преобразованы из строкового типа в числовой

```
>>> name = input('Ваше имя:')
Ваше имя:Чак
>>> print(name)
Чак
>>> apple = input('Количество:')
Количество: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
```

Заглянем внутрь строки

- Мы можем добраться до любого отдельного символа внутри строки, указав в квадратных скобках индекс этого символа
- Значение индекса должно быть целым числом и начинаться с нуля
- Значение индекса также может быть выражением, которое можно вычислить

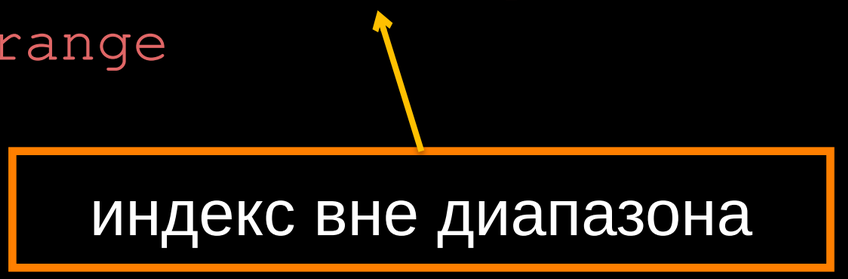
b	a	n	a	n	a
0	1	2	3	4	5

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

Символ за пределами строки

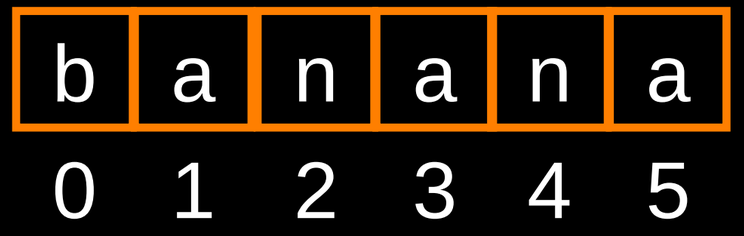
- В Пайтон вы получите ошибку, попытавшись добраться до элемента за пределами конца строки (несуществующий индекс)
- Будьте внимательны при построении значений индексов и срезов

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



У строки есть длина

Встроенная функция len() позволяет нам узнать длину строки




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

Функция len()

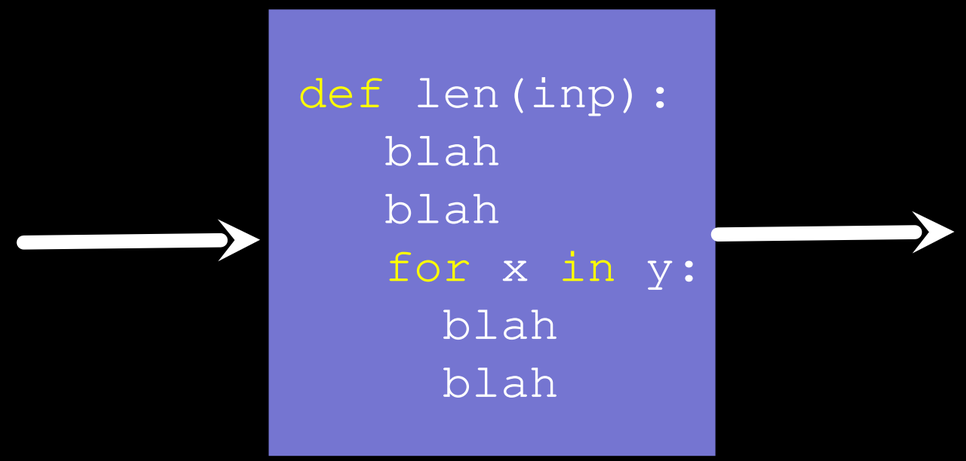
```
>>> fruit = 'banana'
>>> x = len(fruit)
>>> print(x)
6
```

Функция представляет собой некий сохраненный код, который мы используем. Функция принимает входные данные и возвращает вывод/результат.

Функция len()

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

Функция представляет собой некий сохраненный код, который мы используем. Функция принимает входные данные и возвращает вывод/результат.



Пошаговый перебор элементов строки

Используя оператор while, переменную цикла и функцию len(), мы можем создать цикл для доступа к каждой отдельной букве в строке

```
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
```

Пошаговый перебор элементов строки

- Цикл со счетчиком, использующий оператор for, - намного элегантнее
- Цикл с for берет на себя управление переменной цикла

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

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

Пошаговый перебор элементов строки

- Цикл со счетчиком, использующий оператор for, - намного элегантнее
- Цикл с for берет на себя управление переменной цикла

```
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
```

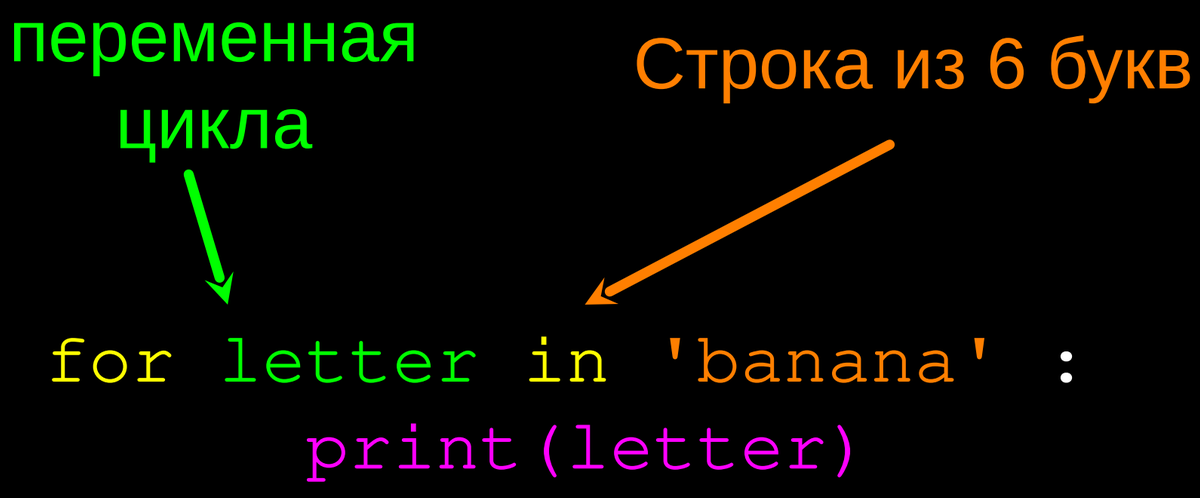
Перебор и подсчет элементов

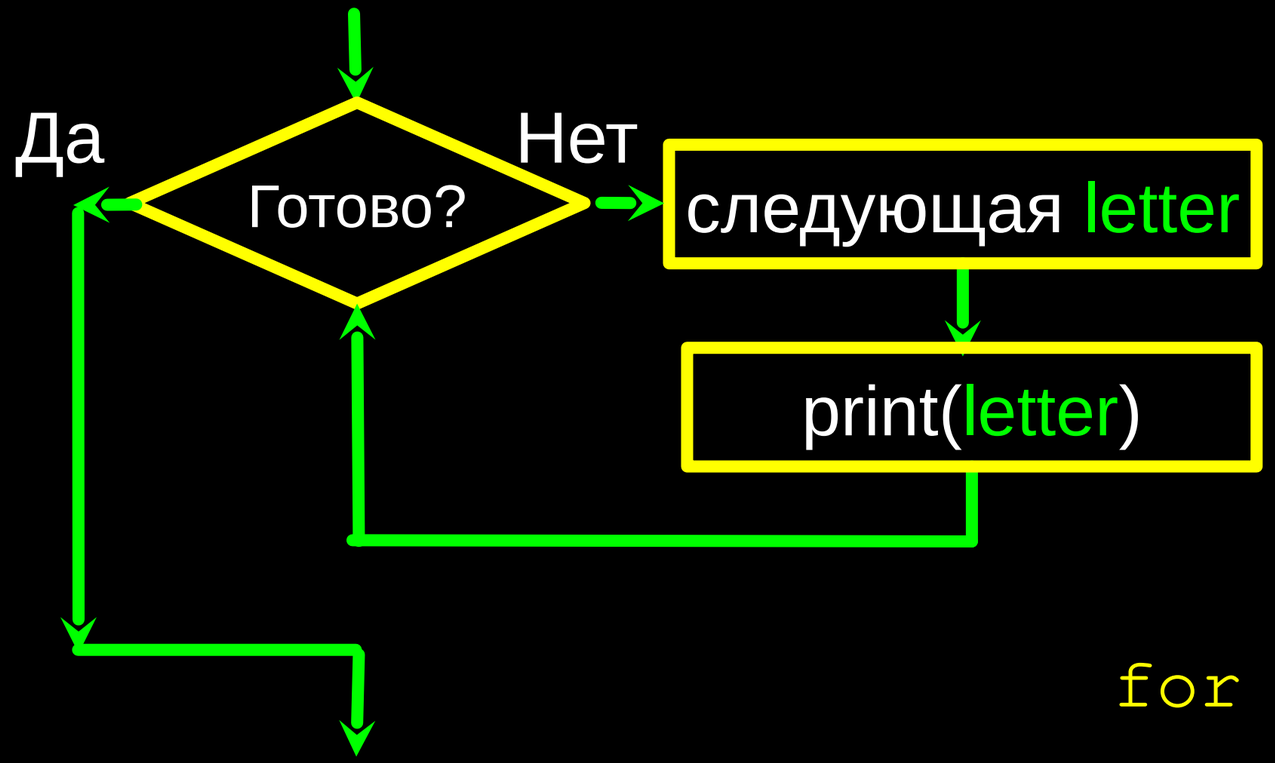
Перед вами простой цикл, который перебирает каждую букву в строке и подсчитывает сколько раз по ходу цикла встречается буква 'a'

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

Присмотримся к оператору in

- Переменная цикла «перемещается» по последовательности (упорядоченному набору)
- Блок (тело) кода выполняется однократно для каждого значения в последовательности
- Переменная цикла поочередно перемещается по всем значениям в последовательности





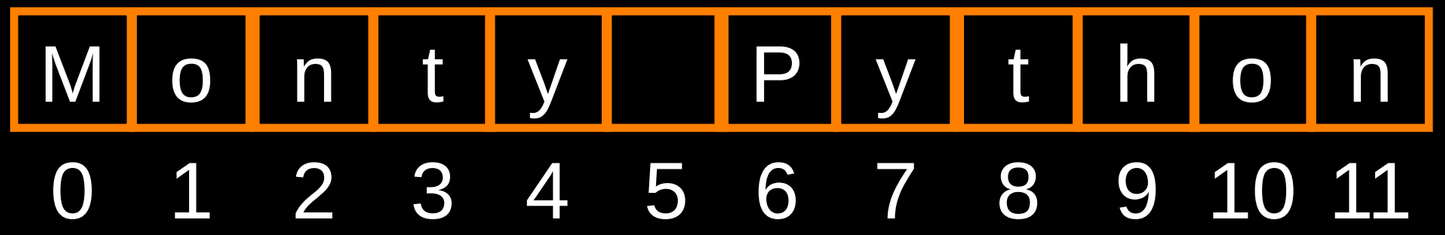
b	a	n	a	n	a

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

Переменная цикла «перемещается» по строке, блок (тело) кода выполняется однократно для каждого значения в последовательности

Ещё операции со строками

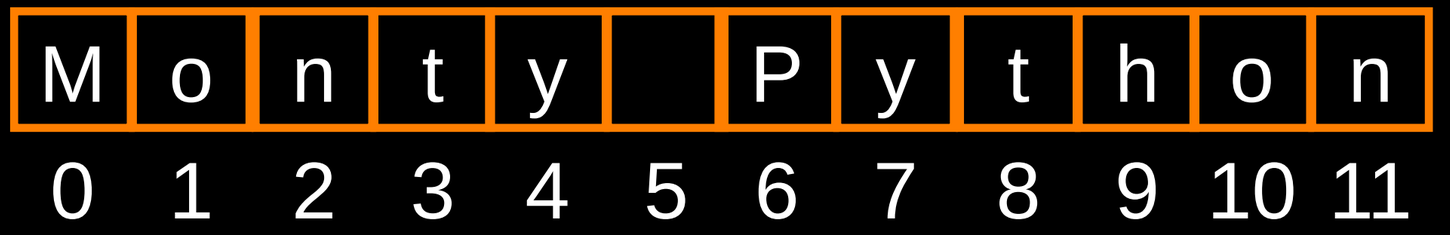
Срез строки



- Мы также можем посмотреть на любой непрерывный кусок строки, используя оператор двоеточие :
- Второе число указывает конец среза, но не включается в него. Срез читается как: «от и до, но не включая»
- Если второе число превышает длину строки, срез будет взят до конца строки

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

Срез строки



Если при извлечении среза мы опускаем первое или последнее число, подразумевается, что пропуск — это начало или конец строки соответственно

```
>>> s = 'Monty Python'
>>> print(s[:2])
Mo
>>> print(s[8:])
thon
>>> print(s[:])
Monty Python
```

Объединение строк

Когда оператор + применяется к строкам, он выполняет функцию «объединения»

```
>>> a = 'Привет'
>>> b = a + 'тебе'
>>> print(b)
Приветтебе
>>> c = a + ' ' + 'There'
>>> print(c)
Привет тебе
>>>
```

Использование оператора in в качестве логического оператора

- Ключевое слово in можно использовать также и для проверки содержится ли одна строка внутри другой
- Выражение с in — это логическое выражение, возвращающее True (Правда) или False (Ложь). Может быть использовано в выражениях с оператором if

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...     print('Найдено!')
...
Найдено!
>>>
```

Сравнение строк

```
if word == 'банан':
    print('Верно, бананы.')

if word < 'банан':
    print('Ваше слово,' + word + ', идет перед бананом.')
elif word > 'банан':
    print('Ваше слово,' + word + ', идет после банана.')
else:
    print('Верно, бананы.')
```

Библиотека для обработки строк

- В Пайтон имеется ряд строковых функций, которые находятся в библиотеке для обработки строк
- Эти функции уже встроены в каждую строку, мы вызываем их, добавляя функцию к строковой переменной
- Эти функции не изменяют исходную строку, а возвращают новую с заданными изменениями

```
>>> greet = 'Привет, Боб'
>>> zap = greet.lower()
>>> print(zap)
привет, боб
>>> print(greet)
Привет, Боб
>>> print('Привет
Всем'.lower())
привет всем
>>>
```

```
>>> stuff = 'Привет, Мир!'
>>> type(stuff)
<class 'str'>
>>> dir(stuff)
['capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
'zfill']
```

https://docs.python.org/3/library/stdtypes.html#string-methods

`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 an ASCII space). The original string is returned if *width* is less than or equal to `len(s)`.

`str.`**`rpartition`**(*sep*)

Split the string at the last occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing two empty strings, followed by the string itself.

`str.`**`rsplit`**(*sep=None*, *maxsplit=-1*)

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done, the *rightmost* ones. If *sep* is not specified or `None`, any whitespace string is a separator. Except for splitting from the right, `rsplit()` behaves like `split()` which is described in detail below.

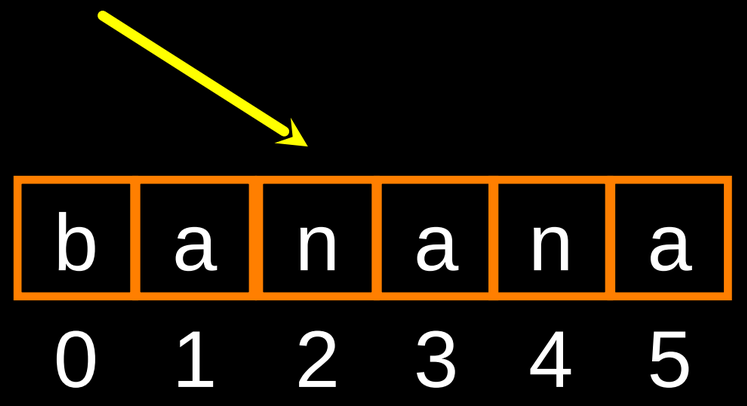
Библиотека для обработки строк

```
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

- Мы используем функцию find() для поиска подстроки внутри другой строки
- find() находит первое появление подстроки
- Если подстрока не найдена, find() возвращает -1
- Помните, что стартовая позиция начинается с 0




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

Преобразование в ЗАГЛАВНЫЕ буквы

- Можно создать копию строки в нижнем регистре или в верхнем регистре
- Прежде чем искать подстроку с помощью find(), лучше преобразовать строку в нижний регистр, чтобы искать вне зависимости от регистра

```
>>> greet = 'Привет, Боб'
>>> nnn = greet.upper()
>>> print(nnn)
ПРИВЕТ, БОБ
>>> www = greet.lower()
>>> print(www)
привет боб
>>>
```

Поиск и замена

- Функция replace() похожа на операцию «поиск и замена» в текстовом редакторе
- Она заменяет содержимое каждой встреченной искомой строки на содержимое заменяющей строки

```
>>> greet = 'Привет, Марина'
>>> nstr =
greet.replace('Марина','Карина')
>>> print(nstr)
Привет, Карина
>>> nstr = greet.replace('р','Х')
>>> print(nstr)
ПХивет КаХина
>>>
```

Удаление пробелов

- Иногда бывает нужно удалить пробелы в начале и/или в конце строки
- lstrip() и rstrip() удаляют пробелы слева и справа соответственно
- strip() одновременно удаляет пробелы в начале и в конце строки

```
>>> greet = '   Привет, Боб  '
>>> greet.lstrip()
'Привет, Боб  '
>>> greet.rstrip()
'   Привет, Боб'
>>> greet.strip()
'Привет, Боб'
>>>
```

Префиксы

```
>>> line = 'Хорошего тебе дня!'
>>> line.startswith('Хорошего')
True
>>> line.startswith('x')
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
```

Два вида строк

```
Python 2.7.10
>>> x = '이광춘'
>>> type(x)
<type 'str'>
>>> x = u'이광춘'
>>> type(x)
<type 'unicode'>
>>>
```

```
Python 3.5.1
>>> x = '이광춘'
>>> type(x)
<class 'str'>
>>> x = u'이광춘'
>>> type(x)
<class 'str'>
>>>
```

В Пайтон 3 все строки — последовательность Юникод-символов

Резюме

- Строковой тип
- Чтение/Преобразование
- Индексация []
- Срез строки [2:4]
- Перебор элементов строки с помощью for и while
- Объединение строк с помощью +
- Операции со строками
- Библиотека операций со строками
- Сравнение строк
- Поиск внутри строк
- Замена текста
- Удаление пробелов

Авторы / Благодарности

Исходная разработка: Чарльз Северанс, Школа Информации Мичиганского Университета.

Перевод выполнила Фомкина Виолетта.

… Добавьте сюда новых авторов и переводчиков

...