



python

Лекция 7

Строки



Строки

Строки — это последовательности символов. В Python строки относятся к неизменяемому типу данных.

Литералы строк объявляются как текст, заключенный в одинарные, двойные или даже тройные кавычки.

Как и в случае списков, так и в случае строк позиция отдельного символа отсчитывается от начала строки.

Начало строки имеет индекс 0. Обращение к отдельному элементу строки имеет такой же синтаксис, как и к элементу списка.



Как создать литералы строк в Python

Для создания литералов строк в Python достаточно указать текстовое содержимое в кавычках.

```
text_one = "Hello "  
text_two = 'word '  
text_three = '''and all'''  
  
print(text_one)  
print(text_two)  
print(text_three)
```

Как видите, строковые литералы могут быть созданы с использованием кавычек разного стиля.



Схожесть списков и строк

Строки обладают **частичной** функциональностью списков. А именно:

- 1) Как и списки, так и строки можно складывать.
- 2) Как и списки, строки можно умножать на целое число.
- 3) Как и в случае списков, доступ к элементу строки для чтения возможен по индексу. Так же работают срезы.

(Внимание! Изменить символ по индексу НЕЛЬЗЯ!)

- 4) По строкам также можно пройти с помощью цикла **for**. Также можно проверить вхождение одной строки в другую с помощью оператора **in**.



Сложение строк - конкатенация

Если к строке прибавить другую строку, то результатом будет строка, равная первой строке, дополненной справа второй строке. Процесс сложения двух строк называется «конкатенация строк».

```
text_one = "Hello "  
text_two = 'word '  
text_three = text_one + text_two  
print(text_three)
```

Если строку умножить на целое число, то результатом будет строка, созданная повторением базовой строки, на заданное числом количество раз.

```
text_one = "Hello "  
text_three = text_one * 4  
print(text_three)
```



Проход по строке с помощью цикла **for**

По строке можно пройти циклом **for**. В таком случае переменная цикла по очереди примет значение каждого символа строки. Например, такой код выведет последовательно символы заданной строки.

```
text_one = "Hello "  
  
for element in text_one:  
    print(element)
```



Получение длины строки и работа с индексами

Как и в случае списков, длину строки можно получить с помощью оператора **len**.

Доступ к подстроке можно получить с помощью срезов. Опять же, как и со списками.

```
text_one = "Hello "  
print(len(text_one))  
  
sub_text = text_one[1:4]  
print(sub_text)
```

Получение длины строки

Получение строки из среза



Строки — неизменяемый тип

Не забывайте, что строка — неизменяемый тип данных. Т.е. получить элемент по индексу вы можете, а изменить нет. Например, такая строка работать не будет:

```
text_one = "Hello "
```

```
text_one[1]="o"
```



**Работать не
будет!!!**



Методы характерные для строк

Изменение строк

Название метода	Описание
<code>upper()</code>	Переводит строку в верхний регистр
<code>lower()</code>	Переводит строку в нижний регистр
<code>swapcase()</code>	Переводит символы нижнего регистра в верхний, а верхнего в нижний
<code>title()</code>	Первую букву каждого слова переводит в верхний регистр, а все остальные в нижний
<code>ljust(width, fillchar=" ")</code>	Делает длину строки не меньшей <code>width</code> , по необходимости заполняя последние символы символом <code>fillchar</code>
<code>rjust(width, fillchar=" ")</code>	Делает длину строки не меньшей <code>width</code> , по необходимости заполняя первые символы символом <code>fillchar</code>
<code>format(*args,**kwargs)</code>	Выполняет форматирование строки
....	И еще много других

Как вызывать такие методы? Укажите имя переменной поставьте точку и напишите имя метода.



Пример использования методов строк (изменение строк)

```
text_one = "Hello "
```

```
text_one = text_one.upper() ← Перевод строки в верхний  
регистр
```

```
print(text_one)
```

```
text_two = "Hello world"
```

```
text_two = text_two.title() ← Каждое слово с большой  
буквы
```

```
print(text_two)
```



Методы характерные для строк. Поиск и проверка.

Название метода	Описание
<code>find(str, [start],[end])</code>	Поиск подстроки (str) в строке в диапазоне индексов от start до end. Возвращает номер первого вхождения или -1
<code>isdigit()</code>	Покажет, состоит ли строка из цифр
<code>isalpha()</code>	Покажет, состоит ли строка из букв
<code>islower()</code>	Покажет, состоит ли строка из символов в нижнем регистре
<code>isupper()</code>	Покажет, состоит ли строка из символов в верхнем регистре
<code>startswith(str)</code>	Покажет, начинается ли строка с шаблона str
<code>endswith(str)</code>	Покажет, заканчивается ли строка S шаблоном str
<code>ord(символ)</code>	Переводит символ в его код ASCII
<code>chr(число)</code>	Переводит код ASCII в символ



Пример использования методов строк(поиск и проверка)

```
text = "Happy new 2017 year"
```

```
digit = 0
```

Проходим по каждому символу строки

```
for l in text:
```

```
    if l.isdigit():  
        digit += 1
```

Проверка символа на то, что он равен цифре

```
print(digit)
```

В этом примере программа считает, сколько цифр содержится в строке.



Разбиение строк на элементы

Можно ли разбить строку на подстроки?

Да. Для этого нужно использовать метод **split**([разделитель]).

Результатом разбиения строки будет список строк.

```
text = "123,345,678"
```

```
text_part = text.split(",")
```

```
for num in text_part:  
    print(num)
```

Разбиение строки по разделителю

« , »

Цикл для вывода полученного
списка на экран

В этом примере строка будет разбита на части по разделителю « , ».



Замена части подстроки

Можно ли заменить в строке одну подстроку другой?

Да. Для этого нужно использовать метод

replace(„что_заменить“, „ чем заменить “).

```
text = "Hello world"
```

```
text = text.replace("world", "Java")
```

```
print(text)
```



Замена подстроки «world» на подстроку «Java»



Что такое ASCII или UTF-8 код символа

**Я слышал, что у символа есть его код.
Можно подробнее об этом?**

Действительно, каждый символ обладает своим кодом. Дело в том, что компьютер «ничего не знает» о символах, которые мы используем для письма. И для того, чтобы обучить ПК нашим символам, пошли на следующий шаг: каждому символу поставили в соответствие цифру, которая и является его кодом.

Так что с точки зрения ПК наш текст - это последовательность цифр.


Первая такая таблица с соответствиями символа и числа, которое ему соответствует, называлась ASCII кодом символов. Однако, по прошествии времени, она была заменена на UTF кодировку.



Пример получения кодов символов

```
text = "Hello world"

for letter in text:
    print(str(ord(letter))+" ")
```


**Получение кода
символа**

Эта программа выведет на экран коды символов, из которых состоит исходная строка.



Форматирование строк с помощью **format**

Когда нужно выровнять строки слева или справа, или округлить вывод числа до нужного знака после запятой, используется метод **format**.

Формат его использования:

„Строка форматирования“.format(список параметров)

„Строка форматирования“ - обычная строка, в которой встречаются символы {}, внутри которых могут встречаться цифры (тогда они обозначают порядковый номер) или имена (тогда это именованные параметры). Буквы и цифры в {} управляют выводом в строку параметров заданных в методе format.



Пример использования метода **format**

**Поля для подстановки
параметров**



```
text = "Hello, I am {}. I learn{}".format("Alexander", "Python")
```



```
print(text)
```

**Строка
форматирования**



**Параметры для
подстановки**

Результатом будет строка «Hello, I am Alexander. I learn Python». Т.е. параметры для подстановки были подставлены вместо {}.



Пример с использованием номера поля

Поля для подстановки параметров



```
text = "Hello, I am {0}. And my name is {0}".format ("Alexander")
```

```
print(text)
```

**Строка
форматирования**

Параметры для подстановки



Так как теперь поля для подстановки указывают на номер поля, то на оба места будет поставлено слово «Alexander».



Пример с использованием именованных полей

Поля с именами для подстановки параметров



```
text = "Hello, I am {name}. I am {age} years old".format(name = "Alexander", age = 36)
```



Параметры для подстановки

Так как теперь поля для подстановки указывают на имя поля, то вместо первого поля для подстановки будет взято «Alexander» , а вместо второго поля будет подставлено значение 36.



Спецификаторы формата для полей подстановки

Для более тонкого управления опциями форматирования в полях для подстановки могут использоваться спецификаторы формата.

Спецификаторы формата ставятся после символа «:» в поле подстановки. Они бывают общего назначения и специализированные.

За этим символом следует:

Символ заполнитель — показывает каким символом будет дополняться значение.

Символ выравнивания:

- < - По левому краю
- > - По правому краю
- ^ - По центру

Минимальная и максимальная ширина поля вывода, разделенные точкой.



Специализированная версия для чисел

С помощью спецификаторов формата можно управлять точностью округления вещественных чисел. Для этого используйте спецификатор формата вида:

`{:ширина поля . количество_знаков_после_запятой}`

После количества_знаков можете указать символ, который указывает на тип вывода.

- E, e - число в экспоненциальной форме
- f - число в обычной форме
- G, g - автоматический вывод



Пример использования спецификатора формата для округления вещественного числа

```
import math
```

```
text = "Pi = {:.4}".format(math.pi)
```

```
print(text)
```



**Число будет округленно до 4-х знаков
после запятой**

Бывают ли еще спецификаторы для других типов данных?

Да. Бывают для целых чисел, строк и т.д.

Где о них при необходимости прочесть?

Например, в рекомендованной литературе.



Список использованной литературы

- 1) Лутц М. Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011 - 126 - 139 с.
- 2) Саммерфилд М. Программирование на Python 3. Подробное руководство. - Пер. с англ. - СПб.:Символ-Плюс, 2009. - 84 - 117 с.



Домашнее задание

- 1) Напишите программу, которая посчитает сколько букв «b» в введенной строке текста.
- 2) Считайте строку, которая будет представлять имя человека, введенное с клавиатуры. Проверьте эту строку на валидность. Имеется в виду, что например, в имени человека не может быть цифр, оно должно начинаться с большой буквы, за которой должны следовать маленькие.
- 3) Напишите программу, которая вычислит сумму всех кодов символов строки.
- 4) Выведите на экран 10 строк со значением числа π . В первой строке должно быть 2 знака после запятой, во второй 3 и так далее.
- 5) Вводится строка из слов, разделенных пробелами. Найти самое длинное слово и вывести его на экран.



Дополнительное домашнее задание.

1) Вовочка сидя на уроке писал подряд одинаковые слова (слово может состоять из одной буквы). Когда Марья Ивановна забрала у него тетрадь, там была одна строка текста. Напишите программу, которая определит самое короткое слово из написанных Вовочкой. Например:

ааааааа — Вовочка писал слово - «а»

ititititit — Вовочка писал слово - «it»

catcatcatcat — Вовочка писал слово - «cat»

2) Напишите программу для очистки текста от html-тэгов. Больше о html-тэгах <https://html5book.ru/html-tags/>

Также необходимо учесть пару особенностей:

- помимо одинарных тэгов, есть парные тэги, например `<div> </div>`, т.е. первый тэг открывающий, а второй закрывающий.

- тэг внутри себя, может содержать кучу доп. информации.

Например `<div id="rcnt" style="clear:both;position:relative;zoom:1">`