Язык программирования Python

05 Строки

Текстовые данные

До этого мы работали с числами, целыми и вещественными, но как нам работать текстовыми данными? Для этого есть строки.

Строка (str) — это тип данных, с помощью которого мы можем работать с текстом.

Вспомним нашу первую программу hello world

```
>>> print('hello world')
```

Здесь ‘hello world’ — это строка. Убедимся в этом.

```
>>> type('hello world')
<class 'str'>
```

С помощью type(obj) мы можем узнавать тип объекта (obj — объект, тип которого необходимо узнать).

Строка

Строки могут хранится в переменных точно так же, как и числа.

```
>>> my_message = ‘hi, my name is Petya’
>>> my_message
‘hi, my name is Petya’
```

Теперь в переменной my_message содержится значение ‘hi, my name is Petya’, и мы можем подставлять эту переменную вместо написания литерала ‘hi, my name is Petya’.

Кавычки

В предыдущих примерах мы брали строки в одинарные кавычки (‘hello world’), но также допустимо использование двойных кавычек (“hello world”).

Между строками ‘hello world’ и “hello world” нет никакой разницы.

Используйте те кавычки, которые вам по душе.

Печать кавычек

Возможны такие ситуации, когда вам необходимо использовать строку, внутри которой есть кавычки. Например, в английском языке очень часто используется одинарная кавычка (he’s a teacher), также может потребоваться выделение названия, предположим, книги в двойные кавычки (роман «Гарри Поттер и философский камень»).

Если мы введем ‘he’s a teacher’, то интерпретатор будет видеть строку ‘he’ и не будет понимать, что делать с оставшейся частью s a teacher’.

Для решения этой проблемы можно поступить следующим образом:

```
>>> "he's a teacher"
"he's a teacher"
>>> 'роман "Гарри Поттер и философский камень"`
'роман "Гарри Поттер и философский камень"'
```

Печать кавычек

Но существует и другой способ — использование escape-последовательностей, или экранированных последовательностей.

Если мы не хотим изменять своим принципам использования кавычек, к которым привыкли, или в строке должны содержаться и те, и другие кавычки, то можно воспользоваться следующей последовательностью \' или \". При печати обратный слеш отображаться не будет.

```
>>> 'he\'s a teacher'
'he's a teacher'
>>> "роман \"Гарри Поттер и философский камень \""
"роман "Гарри Поттер и философский камень ""
```

Существуют и другие экранированные последовательности.

Escape-последовательности

Рассмотрим основные escape-последовательности, которые могут нам пригодиться.

- \\ — позволяет напечатать сам символ обратного слеша внутри строки (при выводе с помощью функции print).
- \' — позволяет напечатать одинарную кавычку (апостроф)
- \" — позволяет напечатать двойную кавычку
- \n — перевод строки (при выводе с помощью функции print).

```
>>> print('hello\nworld')
'hello
world‘
```

- \t — табуляция (при выводе с помощью функции print).

```
>>> print('hello\t\t\t\tworld')
hello                                world
```

Строковые литералы в несколько строк

Строковые литералы могут занимать несколько строк. Для этого можно использовать три кавычки подряд ‘’’…’’’ или “””…”””.

```
>>> msg1 = '''
str1
str2
str3
'''
>>> msg1
'\nstr1\nstr2\nstr3\n'
>>> print(msg1)

str1
str2
str3

>>>
```

```
>>> msg2 = '''\
str1
str2
str3
'''
>>> msg2
'str1\nstr2\nstr3\n'
>>> print(msg2)
str1
str2
str3

>>>
```

Концы строк включаются в нашу строку, избежать этого можно при помощи escape-последовательности \ в конце строки.

Длина строки

Мы можем воспринимать нашу строку как последовательность символов и можем получить доступ к определенным символам этой строки.

Давайте для начала попробуем узнать количество символов в наше строке. Для этого используется функция len (от англ. length — длина).

```
>>> msg = ‘hello world’
>>> len(msg)
11
```

Функция len насчитала 11 символов в нашей строке, можете пересчитать самостоятельно.

Доступ/обращение к символам строки

Для получения отдельного символа или символов строки надо использовать квадратные скобки после нашей строки. У каждого символа в строке есть индекс, но начинается он с нуля, то есть у первого по счету символа будет индекс 0, если строка состоит из 11 символов, то индекс последнего символа 10.

my_string[start]

```
>>> msg = 'hello world'
>>> len(msg)
11
>>> msg
'hello world'
>>> msg[0]
'h'
>>> msg[10]
'd'
>>> msg[11]
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    msg[11]
IndexError: string index out of range
>>>
```

Срез строки (slicing)

Мы также можем получить доступ сразу к нескольким символам строки.

Для этого внутри квадратных скобок необходимо указать начальный и конечный индексы (символ конечного индекса в выборку не входит).

my_string[start:stop]

Если start == 0, то его можно опустить my_string[:stop]

Если мы хотим взять строку с какого-то символа и до конца, то my_string[start:]

```
>>> msg
'hello world'
>>> msg[0:5]
'hello'
>>>
```

Конкатенация строк

Строки можно складывать, такая операция называется конкатенация. При конкатенации к концу строки, стоящей слева от знака +, прибавляется вторая строка, стоящая справа от знака +. Оба операнда — строки.

```
>>> msg1 = 'hi'
>>> msg2 = 'bye'
>>> msg3 = msg1 + msg2
>>> msg3
'hibye'
>>> msg4 = msg1 + ' ' + msg2
>>> msg4
'hi bye'
```

Повторение строк

Операция повторения позволяет повторять строку. Операция представлена символом умножения. Один операнд должен быть строкой, другой — целым числом.

```
>>> msg1 = 'hi'
>>> msg2 = 'bye'
>>> msg3 = msg1 * 10
>>> msg3
'hihihihihihihihihihi'
```

```
>>> msg4 = msg1 * msg2
Traceback (most recent call last):
  File "<pyshell#54>", line 1, in <module>
    msg4 = msg1 * msg2
TypeError: can't multiply sequence by non-int of type 'str'
>>>
```

Итоги

Мы познакомились с такими понятиями, как строка, строковый литерал, кавычки, экранированные символы, индекс, срез строки, конкатенация строк, повторение строк,

а также попрактиковались в работе с интерпретатором в интерактивном режиме.