Введение в строки. Базовые операции над строками

Сегодня мы с вами познакомимся с еще одним типом данных –строками.

Строки в Python задаются очень просто: или в двойных кавычках:

`s1 = "Панда"` или в одинарных (апострофах): `s2 = 'Panda'`

Всегда, когда задаются строки, не забывайте про кавычки, если их не поставить: `s2 = Panda`

то Panda будет восприниматься как переменная и возникнет ошибка.

В Python есть еще один способ определения многострочных строк. Для этого используются тройные кавычки (одинарные или двойные, неважно) и в них прописывается текст, например, так:

```
text = '''Я Python бы выучил толь что
есть популярные курсы.
Много хороших курсов!'''
```

запомните, что для перехода на новую строку используется спецсимвол, который записывается в виде ‘\n’. Если записать просто ‘n’ – это будет символ латинской буквы n, а при добавлении слеша – превращается в символ переноса строки.

Далее, строка может вообще не содержать ни одного символа: `a = ""`

Получаем пустую строку. Но если добавить хотя бы один символ, даже если это будет пробел: `a = " "`

то имеем уже не пустую строку, в данном случае содержащей символ пробела.

Базовые операции над строками

Давайте посмотрим, какие базовые операции можно выполнять со строками в Python. Например, мы хотим соединить две строки между собой:

```
s1 = "Я люблю"
s2 = "язык Python"
```

Это можно сделать с помощью оператора +, который в случае со строками выполняет их объединение (конкатенацию):

```
s3 = s1 + s2
print(s3)
```

Но мы бы хотели добавить пробел между словами. Сделаем это так: `s3 = s1 + " " + s2`

С помощью первого оператора + добавляем пробел к первой строке, а затем, вторым оператором + добавляем вторую строку s2.

Но при использовании оператора конкатенации следует быть осторожным – он объединяет строки между собой. Например, команда: `s3 = s1 + 5`

приведет к ошибке, так как операнд справа является числом, а не строкой. Если нам все же необходимо соединить строку с числом, то предварительно число нужно преобразовать в строку. Сделать это можно с помощью специальной функции str(): `s3 = s1 + str(5)`

Функция str() выполняет преобразование в строки разные типы данных, не только числа, например, можно указать булевое значение: `str(True)`

а также другие типы данных, о которых мы еще с вами будем говорить.

Следующий оператор *, применительно к строкам, выполняет их дублирование, указанное число раз: `"xa " * 5`

Причем, здесь мы должны указывать именно целое число, для вещественных получим ошибку: `"xa " * 5.5`

И это понятно, так как продублировать строку 5,5 раз нельзя.

Следующая функция len() возвращает длину строки (число символов в строке):

```
a = "hello"
len(a)
```

Для пустой строки получим значение 0: `len("")`

И, как видите, этой функции можно передавать или переменную на строку, или непосредственно записывать строки: `len("Python")`

Следующий оператор in позволяет проверять наличие подстроки в строке, например:

```
'ab' in "abracadabra"
'abc' in "abracadabra"
```

Следующая важная группа операторов – сравнения строк. В самом простом случае, строки можно сравнивать на равенство:

`a == "hello"` Но сравнение: `a == "Hello"`

вернет False, так как большая буква H и малая h – это два разных символа. Для сравнения на неравенство используем оператор не равно:

```
a != "hello"
a != "hello "
```

Также смотрите строка "hello" (без пробела) и строка "hello " (с пробелом) – это две разные строки и они не равны между собой.

Наконец, строки можно сравнивать на больше и меньше, например, кот больше, чем кит с точки зрения строк: `'кот' > 'кит'`

Почему так? Все просто. Здесь используется лексикографический порядок сравнения. Сначала берутся первые символы (они равны), затем переходим ко вторым символам. По алфавиту сначала идет символ ‘и’, а потом – символ ‘о’, поэтому ‘о’ больше, чем ‘и’. Как только встретились не совпадающие символы, сравнение завершается и последующие символы строк игнорируются.

Если взять равные строки: `'кот' > 'кот'`

то получим False, так как ни один символ не больше соответствующего другого из второй строки. Но, добавив пробел в первую строку: `'кот ' > 'кот'`

получим значение True, так как при всех прочих равных условиях больше считается более длинная строка. Наконец, если у первой строки первую букву сделать заглавной: `'Кот ' > 'кот'`

то получим False. Почему? Дело в том, что каждый символ в компьютере связан с определенным числом – кодом, в соответствии с кодовой таблицей. Например, в таблице ASCII мы видим, что сначала идут символы заглавных букв, а затем – прописных. Поэтому коды больших букв меньше соответствующих кодов малых букв.

Конечно, в Python используется немного другая кодировка UTF-8, но в ней этот принцип сохраняется. Мы можем легко посмотреть код любого символа с помощью функции ord():

```
ord('К')
ord('к')
```

И, как видите, для буквы ‘К’ код меньше, чем для ‘к’.

Итак, из этого занятия вам нужно запомнить, как задавать обычные и многострочные строки. Что из себя представляет символ переноса строки. Знать базовые операции со строками:

- + (конкатенация) – соединение строк;
- * (дублирование) – размножение строкового фрагмента;
- str() – функция для преобразования аргумента в строковое представление;

- len() – вычисление длины строки;
- in – оператор для проверки вхождения подстроки в строку;
- операторы сравнения: == != > <
- ord() – определение кода символа.