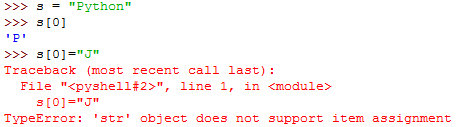
**Строки**

Строки являются упорядоченными последовательностями символов. Длина строки ограничена лишь объемом оперативной памяти компьютера. Как и все последовательности, строки поддерживают обращение к элементу по индексу, получение среза, конкатенацию (оператор **+**), повторение (оператор **\***), проверку на вхождение (оператор **in**).

Кроме того, строки относятся к неизменяемым типам данных. Поэтому практически все строковые методы в качестве значения возвращают новую строку. При использовании небольших строк это не приводит к каким-либо проблемам, но при работе с большими строками можно столкнуться с проблемой нехватки памяти. Можно получить символ по индексу, но изменить его нельзя.



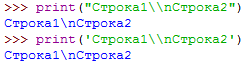
В некоторых языках программирования концом строки является нулевой символ. В языке **Python** нулевой символ может быть расположен внутри строки:



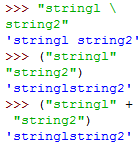
Создать строку можно следующими способами:

* с помощью функции **str()**. Если в качестве параметра передать объект, то функция возвратит строковое представление этого объекта. Если параметр не указан, то возвращается пустая строка.
* указав строку между апострофами или двойными кавычками:

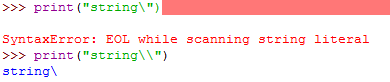
В языке **Python** никакого отличия между строкой в апострофах и строкой в кавычках нет. Это одно и то же. Если строка содержит кавычки, то ее лучше заключить в апострофы и наоборот. Все специальные символы в таких строках интерпретируются. Например, последовательность символов **\n** преобразуется в символ новой строки. Чтобы специальный символ вводился как есть, его необходимо экранировать с помощью слеша:



Следует также заметить, что заключить объект в одинарные кавычки (или апострофы) на нескольких строках нельзя. Переход на новую строку вызовет синтаксическую ошибку. Чтобы расположить объект на нескольких строках, следует перед символом перевода строки указать символ \ (после \ не должно быть никаких символов), поместить две строки внутри скобок (неявная конкатенация строк) или использовать конкатенацию внутри скобок (явная конкатенация строк):

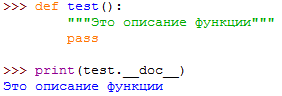


Кроме того, если в конце строки расположен символ \, то его необходимо экранировать, иначе будет выведено сообщение об ошибке: ·



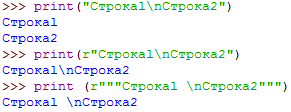
* указав строку между утроенными апострофами или утроенными кавычками. Такие объекты можно разместить на нескольких строках, а также одновременно использовать кавычки и апострофы без необходимости их экранировать. В остальном такие объекты эквивалентны строкам в апострофах и кавычках. Все специальные символы в таких строках интерпретируются.

Если строка не присваивается переменной, то она считается строкой документирования. Такая строка сохраняется в атрибуте \_\_doc\_\_ того объекта, в котором расположена. В качестве примера создадим функцию со строкой документирования, а затем выведем содержимое строки:



Так как выражения внутри таких строк не выполняются, то утроенные кавычки (или утроенные апострофы) очень часто используются для комментирования больших фрагментов кода на этапе отладки программы.

Если перед строкой разместить модификатор **r**, то специальные символы внутри строки выводятся как есть. Например, символ **\n** не будет преобразован в символ перевода строки. Иными словами, он будет считаться последовательностью двух символов: \ и n:



Такие неформатированные строки удобно использовать в шаблонах регулярных выражений, а также при указании пути к файлу или каталогу:



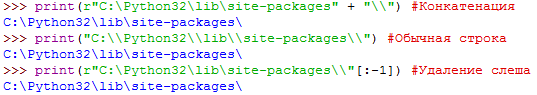
Если модификатор не указать, то все слеши в пути необходимо экранировать:



Если в конце неформатированной строки расположен слеш, то его необходимо экранировать. Однако следует учитывать, что этот слеш будет добавлен в исходную строку. Пример:



Чтобы избавиться от лишнего слеша, можно использовать операцию конкатенации строк, обычные строки или удалить слеш явным образом:



**7.2 Специальные символы**

Специальные символы - это комбинации знаков, обозначающих служебные или непечатаемые символы, которые невозможно вставить обычным способом. Перечислим специальные символы, допустимые внутри строки, перед которой нет модификатора r:

* **\n** перевод строки;
* **\r** возврат каретки;
* **\t** знак табуляции;
* **\v** вертикальная табуляция;
* **\а** звонок;
* **\b** забой;
* **\f** перевод формата;
* **\0** нулевой символ (не является концом строки);
* **\"** кавычка;
* **\'** апостроф;
* **\N** восьмеричное значение N. Например, \74 соответствует символу <;
* **\xN** шестнадцатеричное значение N. Например, \х6a соответствует символу j;
* **\\** обратный слеш;
* **\uxxxx** 16-битный символ **Unicode**. Например, \u043a соответствует русской букве к;
* **\Uxxxxxxxx** 32-битный символ **Unicode**.

Если после слеша не стоит символ, который вместе со слешем интерпретируется как спецсимвол, то слеш сохраняется в составе строки:



Тем не менее, лучше экранировать слеш явным образом:



**7.3 Операции над строками**

Как вы уже знаете, строки относятся к последовательностям. Как и все последовательности, строки поддерживают обращение к элементу по индексу, получение среза, конкатенацию, повторение и проверку на вхождение. Рассмотрим эти операции подробно.

К любому символу строки можно обратиться как к элементу списка. Достаточно указать его индекс в квадратных скобках. Нумерация начинается с нуля:



Если символ, соответствующий указанному индексу, отсутствует в строке, то возбуждается исключение **IndexError**.

В качестве индекса можно указать отрицательное значение. В этом случае смещение будет отсчитываться от конца строки, а точнее, значение вычитается из длины строки, чтобы получить положительный индекс:



Так как строки относятся к неизменяемым типам данных, то изменить символ по индексу нельзя. Чтобы выполнить изменение, можно воспользоваться операцией извлечения среза, которая возвращает указанный фрагмент строки. Формат операции:

**[<Начало>:<Конец>:<Шаг>]**

Все параметры являются необязательными. Если параметр <**Начало**> не указан, то используется значение **0**. Если параметр <**Конец**> не указан, то возвращается фрагмент до конца строки. Следует также заметить, что символ с индексом, указанным в этом параметре, не входит в возвращаемый фрагмент. Если параметр <Шаг> не указан, то используется значение **1**. В качестве значения параметров можно указать отрицательные значения.

Теперь рассмотрим несколько примеров. Сначала получим копию строки:



Теперь выведем символы в обратном порядке:



Заменим первый символ в строке:



Удалим последний символ:



Получим первый символ в строке:



А теперь получим последний символ:



И, наконец, выведем символы с индексами 2, 3 и 4:



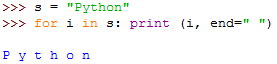
Узнать количество символов в строке позволяет функция **len()**:



Теперь, когда мы знаем количество символов, можно перебрать все символы с помощью цикла **for**:



Так как строки поддерживают итерации, можно просто указать строку в качестве параметра цикла:



Соединить две строки в одну строку позволяет оператор **+**:



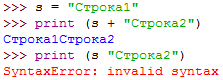
Кроме того, можно выполнить неявную конкатенацию строк. В этом случае две строки указываются рядом без оператора между ними:



Обратите внимание на то, что если между строками указать запятую, то мы получим кортеж, а не строку:



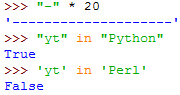
Если соединяются, например, переменная и строка, то следует обязательно указывать символ конкатенации строк, иначе будет выведено сообщение об ошибке:



При необходимости соединить строку с другим типом данных (например, с числом) следует произвести явное преобразование типов с помощью функции **str()**:



Кроме рассмотренных операций, строки поддерживают операцию повторения и проверку на вхождение. Повторить строку указанное количество раз можно с помощью оператора **\***, а выполнить проверку на вхождение фрагмента в строку позволяет оператор **in**:



**7.4. Форматирование строк**

Вместо соединения строк с помощью оператора + лучше использовать форматирование. Данная операция позволяет соединять строку с любым другим типом данных и выполняется быстрее конкатенации.

**ПРИМЕЧАНИЕ**

В последующих версиях **Python** оператор форматирования **%** может быть удален. Вместо этого оператора в новом коде следует использовать метод **format()** , который рассматривается в следующем разделе.

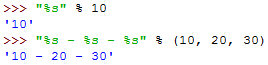
Форматирование имеет следующий синтаксис:

**<Строка специального формата> % <Значения>**

Внутри параметра **<Строка специального формата>** могут быть указаны спецификаторы, имеющие следующий синтаксис:

**% [(<Ключ>)] [<Флаг>] [<Ширина>] [.<Точность>] <Тип преобразования>**

Количество спецификаторов внутри строки должно быть равно количеству элементов в параметре <**Значения**>. Если используется только один спецификатор, то параметр <**Значения**> может содержать одно значение, в противном случае необходимо перечислить значения через занятую внутри круглых скобок, создавая тем самым кортеж. Пример: ·

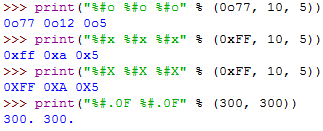


Параметры внутри спецификатора имеют следующий смысл:

* <**Ключ**> ключ словаря. Если задан ключ, то в параметре <**Значения**> необходимо указать словарь, а не кортеж. Пример:



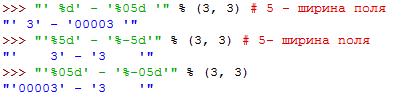
* <**Флаг**>- флаг преобразования. Может содержать следующие значения:
  + **#** для восьмеричных значений добавляет в начало комбинацию символов 0**о**, для шестнадцатеричных значений добавляет комбинацию символов ox (если используется тип х) или 0X (если используется тип X), для вещественных чисел указывает всегда выводить дробную точку, даже если задано значение 0 в параметре <**Точность**>:



* + **0** задает наличие ведущих нулей для числового значения:



* + **-** задает выравнивание по левой границе области. По умолчанию используется выравнивание по правой границе. Если флаг указан одновременно с флагом 0, то действие флага 0 будет отменено. Пример:



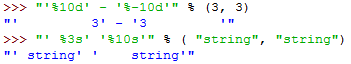
* + **пробел** - вставляет пробел перед положительным числом. Перед отрицательным числом будет стоять минус. Пример:



* + **+** задает обязательный вывод знака, как для отрицательных, так и для положительных чисел. Если флаг **+** указан одновременно с флагом пробел, то действие флага пробел будет отменено. Пример:



* <**Ширина**> минимальная ширина поля. Если строка не помещается в указанную ширину, то значение игнорируется и строка выводится полностью:



Вместо значения можно указать символ "\*" В этом случае значение следует задать внутри кортежа:



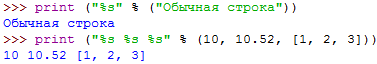
* <**Точность**>- количество знаков после точки для вещественных чисел. Перед этим параметром обязательно должна стоять точка. Пример:



Вместо значения можно указать символ "\*". В этом случае значение следует задать внутри кортежа:



* <**Тип преобразования**>- задает тип преобразования. Параметр является обязательным. В параметре <**Тип преобразования**> могут быть указаны следующие символы:
  + **s** преобразует любой объект в строку с помощью функции **str()**:



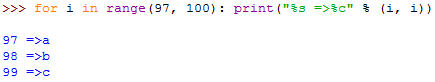
* + **r** преобразует любой объект в строку с помощью функции **repr()**:



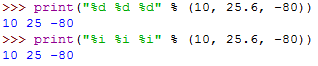
* + **а** преобразует объект в строку с помощью функции **ascii()**:



* + **с** выводит одиночный символ или преобразует числовое значение в символ. В качестве примера выведем числовое значение и соответствующий этому значению символ:



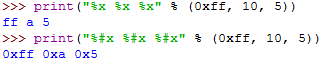
* + **d** и **i** возвращают целую часть числа:



* + **о** восьмеричное значение:



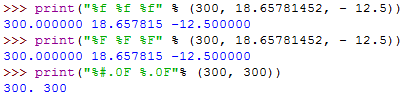
* + **х** шестнадцатеричное значение в нижнем регистре:



* + **X** шестнадцатеричное значение в верхнем регистре:

,

* + **f** и **F** вещественное число в десятичном представлении:



* + **е** вещественное число в экспоненциальной форме (буква "е" в нижнем регистре):



* + **E** вещественное число в экспоненциальной форме (буква "е" в верхнем регистре):



* + **g** эквивалентно **f** или **е** (выбирается более короткая запись числа):



* + **G**- эквивалентно **f** или **Е** (выбирается более короткая запись числа):

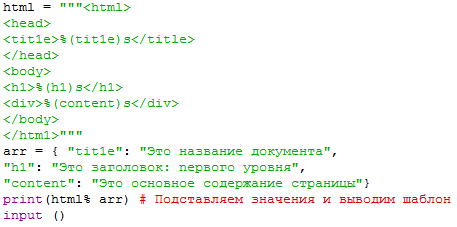


Если внутри строк необходимо использовать символ процента, то этот символ следует удвоить, иначе будет выведено сообщение об ошибке:

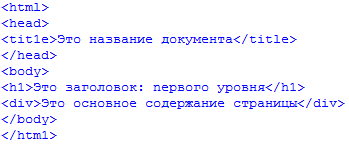


Форматирование строк очень удобно использовать при передаче данных в шаблон **НТМL**-страницы.

Для этого заполняем словарь данными и указываем его справа от символа %, а сам шаблон - слева. Продемонстрируем это на примере.



Результат выполнения:

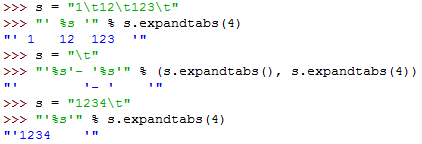


Для форматирования строк можно также использовать следующие методы:

* **expandtabs ( [<Ширина nоля> ] )** - заменяет символ табуляции пробелами таким образом, чтобы общая ширина фрагмента вместе с текстом (расположенным перед символом табуляции) была равна указанной величине. Если параметр не указан, то ширина поля предполагается равной 8 символам. Пример:



В этом примере ширина задана равной четырем символам. Поэтому во фрагменте "1\t" табуляция будет заменена тремя пробелами, во фрагменте "12\t"- двумя пробелами, а во фрагменте "123\t"- одним пробелом. Во всех трех фрагментах ширина будет равна четырем символам. Если перед символом табуляции нет текста или количество символов перед табуляцией равно ширине, то табуляция заменяется указанным количеством пробелов:



Если количество символов перед табуляцией больше ширины, то табуляция заменяется пробелами таким образом, чтобы ширина фрагмента вместе с текстом делилась без остатка на указанную ширину:



Таким образом, если количество символов перед табуляцией больше **4**, но менее **8**, то фрагмент дополняется пробелами до **8** символов. Если количество символов больше **8**, но менее **12**, то фрагмент дополняется пробелами до **12** символов и т. д. Все это справедливо при указании в качестве параметра числа **4**;

* **center (<Ширина> [, <Символ> ])** - производит выравнивание строки по центру внутри поля, указанной ширины. Если второй параметр не указан, то справа и слева от исходной строки будут добавлены пробелы. Пример:



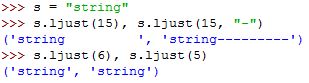
Теперь произведем выравнивание трех фрагментов шириной 15 символов. Первый фрагмент будет выровнен по правому краю, второй - по левому, а третий - по центру:



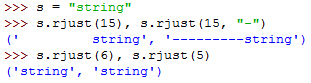
Если количество символов в строке превышает ширину поля, то значение ширины игнорируется и строка возвращается полностью:



* **ljust(<Ширина>[, <Символ>])** - производит выравнивание строки по левому краю внутри поля указанной ширины. Если второй параметр не указан, то справа от исходной строки будут добавлены пробелы. Если количество символов в строке превышает ширину поля, то значение ширины игнорируется и строка возвращается полностью. Пример:



* **rjust (<Ширина> [, <Символ> ])** - производит выравнивание строки по правому краю внутри поля указанной ширины. Если второй параметр не указан, то слева от исходной строки будут добавлены пробелы. Если количество символов в строке превышает ширину поля, то значение ширины игнорируется и строка возвращается полностью. Пример:



* **zfill (<Ширина>)** - производит выравнивание фрагмента по правому краю внутри поля указанной ширины. Слева от фрагмента будут добавлены нули. Если количество символов в строке превышает ширину поля, то значение ширины игнорируется и строка возвращается полностью. Пример:



**7.5. Метод *format()***

Начиная с **Python 2.6**. помимо операции форматирования, строки поддерживают метод **format()**. Метод имеет следующий синтаксис:

**<Строка>= <Строка специального формата>.format(\*args, \*\*kwargs)**

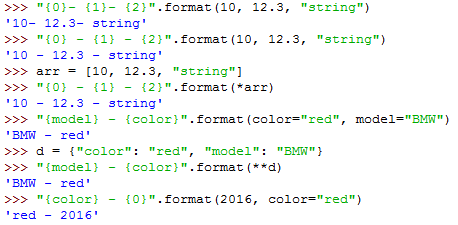
В параметре <Строка специального формата> внутри символов { и } указываются спецификаторы, имеющие следующий синтаксис:

**{[<Поле>] [!<Функция>] [:<Формат>]}**

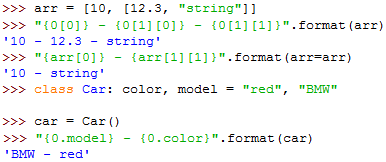
Все символы, расположенные вне фигурных скобок, выводятся без преобразований. Если внутри строки необходимо использовать символы { и }, то эти символы следует удвоить, иначе возбуждается исключение **ValueError**. Пример:



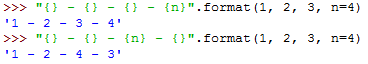
В параметре <**Поле**> можно указать индекс позиции (нумерация начинается с нуля) или ключ. Допустимо комбинировать позиционные и именованные параметры. В этом случае в методе **format()** именованные параметры указываются в самом конце. Пример:



В качестве параметра в методе **format()** можно указать объект. Для доступа к элементам по индексу внутри строки формата применяются квадратные скобки, а для доступа к атрибутам объекта используется точечная нотация:

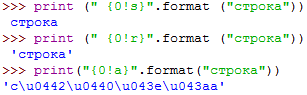


Существует также краткая форма записи, при которой параметр <**Поле**> не указывается. В этом случае скобки без указанного индекса нумеруются слева направо, начиная с нуля:



Параметр <**Функция**> задает функцию, с помощью которой обрабатываются данные перед вставкой в строку. Если указано значение "s", то данные обрабатываются функцией **str()**, если значение "**r**", то функцией **repr()**, а если значение "**а**", то функцией **ascii()**. Если параметр не указан, то для преобразования данных в строку используется функция **str()**.

Пример:



В параметре <**Формат**> указывается значение, имеющее следующий синтаксис:

**[ [<Заполнитель>] <Выравнивание>] [<Знак>] [ #] [0] [<Ширина>] [, ] [.<Точность>] [ <Преобразование>]**

Параметр <**Ширина**> задает минимальную ширину поля. Если строка не помещается в указанную ширину, то значение игнорируется и строка выводится полностью:



Ширину поля можно передать в качестве параметра в методе **format()**. В этом случае вместо числа указывается индекс параметра внутри фигурных скобок:



По умолчанию значение внутри поля выравнивается по правому краю. Управлять выравниванием позволяет параметр <**Выравнивание**>. Можно указать следующие значения:

* **<** по левому краю;
* **>** по правому краю;
* **^** по центру поля.

Пример:



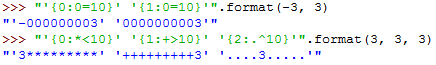
* = знак числа выравнивается по левому краю, а число по правому краю:



Как видно из предыдущего примера, пространство между знаком и числом по умолчанию заполняется пробелами, а знак положительного числа не указывается. Чтобы вместо пробелов пространство заполнялось нулями, необходимо указать нуль перед шириной поля:

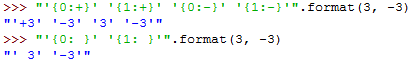


Такого же эффекта можно достичь, указав нуль в параметре <**Заполнитель**>. В этом параметре допускаются и другие символы, которые будут выводиться вместо пробелов:



Управлять выводом знака числа позволяет параметр <**Знак**>. Допустимые значения:

* + задает обязательный вывод знака, как для отрицательных, так и для положительных чисел;
* - вывод знака только для отрицательных чисел (значение по умолчанию);
* пробел вставляет пробел перед положительным числом. Перед отрицательным числом будет стоять минус. Пример:



Для целых чисел в параметре <**Преобразование**> могут быть указаны следующие опции:

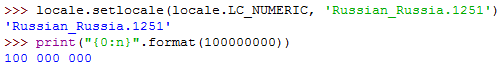
* **b** - двоичное значение:



* **с** - преобразует целое число в соответствующий символ:



* **d** - десятичное значение;
* **n** - аналогично опции **d**, но учитывает настройки локали. Например, выведем большое число с разделением тысячных разрядов пробелом:



Можно также разделить тысячные разряды запятой, указав ее в строке формата:



* **о** - восьмеричное значение:



* **х** - шестнадцатеричное значение в нижнем регистре:



* X - шестнадцатеричное значение в верхнем регистре:



Для вещественных чисел в параметре <**Преобразование**> могут быть указаны следующие опции:

* **f** и **F** - вещественное число в десятичном представлении:



Задать количество знаков после запятой позволяет параметр <**Точность**>:



* е - вещественное число в экспоненциальной форме (буква "е" в нижнем регистре):



* E - вещественное число в экспоненциальной форме (буква "е" в верхнем регистре):



* **g** - эквивалентно **f** или **е** (выбирается более короткая запись числа):



* **n** - аналогично опции **g**, но учитывает настройки локали;
* **G** - эквивалентно **f** или **Е** (выбирается более короткая запись Числа):



* % - умножает число на **100** и добавляет символ процента в конец. Значение отображается в соответствии с опцией **f**. Пример:



**7.6 Функции и методы для работы со строками**

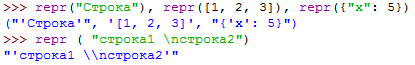
Рассмотрим основные функции для работы со строками:

* **str( [<Объект> ])** - преобразует любой объект в строку". Если параметр не указан, то возвращается пустая строка. Используется функцией **print()** для вывода объектов.

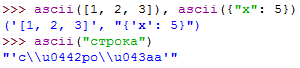
Пример:



* **repr(<Объект>)** - возвращает строковое представление объекта. Используется при выводе объектов в окне **Python Shell** редактора **IDLE**. Пример:

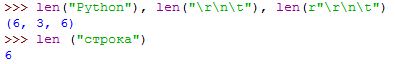


* **ascii (<Объект>)** - возвращает строковое представление объекта. В строке могут быть символы только из кодировки ASCII. Пример:



Английские буквы выводятся как символы, русские как коды.

* **len (<Строка>)** - возвращает количество символов в строке:



Перечислим основные методы:

* **strip ( [<Символы> ])** - удаляет пробельные или указанные символы в начале и конце строки. Пробельными символами считаются: пробел, символ перевода строки (\n), символ возврата каретки (\r), символы горизонтальной (\t) и вертикальной (\v) табуляции:



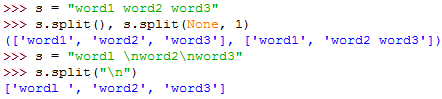
* **lstrip( [<Символы> ])** - удаляет пробельные или указанные символы в начале строки:



* **rstrip( [<Символы> ])** - удаляет пробельные или указанные символы в конце строки:



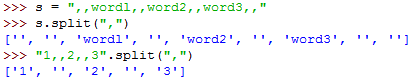
* **split ( [<Разделитель> [, <Лимит> ]])** - разделяет строку на подстроки по указанному разделителю и добавляет их в список. Если первый параметр не указан или имеет значение **None**, то в качестве разделителя используется символ пробела. Если во втором параметре задано число, то в списке будет указанное количество подстрок. Если подстрок больше указанного количества, то список будет содержать ещё один элемент, в котором будет остаток строки. Примеры:



Если в строке содержатся несколько пробелов подряд и разделитель не указан, то пустые элементы не будут добавлены в список:



При использовании другого разделителя могут быть пустые элементы:



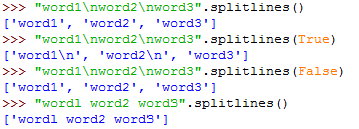
Если разделитель не найден в строке, то список будет состоять из одного элемента, представлявшего исходную строку:



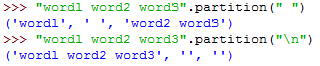
* **rsplit([<Разделитель>[, <Лимит>]])** - метод, аналогичен методу **split()**, но поиск символа-разделителя производится не слева направо, а, наоборот, справа налево. Примеры:



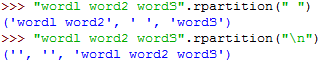
* **splitlines ( [True])** - разделяет строку на подстроки по символу перевода строки (**\n**) и добавляет их в список. Символы новой строки включаются в результат, только если необязательный параметр имеет значение **True**. Если разделитель не найден в строке, то список будет содержать только один элемент. Примеры:



* **partition(<Разделитель>)** - находит первое вхождение символа-разделителя в строку и возвращает кортеж из трех элементов. Первый элемент будет содержать фрагмент, расположенный перед разделителем, второй элемент - символ-разделитель, а третий элемент - фрагмент, расположенный после символа-разделителя. Поиск производится слева направо. Если символ-разделитель не найден, то первый элемент кортежа будет содержать всю строку, а остальные элементы будут пустыми. Пример:



* **rpartition(<Разделитель>)** - метод аналогичен методу **partition()**, но поиск символа разделителя производится не слева направо, а, наоборот, справа налево. Если символ разделитель не найден, то первые два элемента кортежа будут пустыми, а третий элемент будет содержать всю строку. Пример:



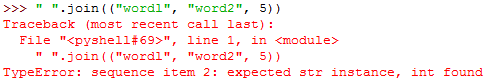
* **join()** - преобразует последовательность в строку. Элементы добавляются через указанный разделитель. Формат метода:

**<Строка> = <Разделитель>.jоin(<Последовательность>)**

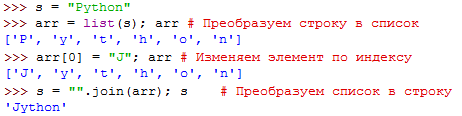
В качестве примера преобразуем список и кортеж в строку:



Обратите внимание на то, что элементы последовательностей должны быть строками, иначе возбуждается исключение **TypeError**:



Как вы уже знаете, строки относятся к неизменяемым типам данных. Если попытаться изменить символ по индексу, то возникнет ошибка. Чтобы изменить символ по индексу можно преобразовать строку в список с помощью функции list(), произвести изменения, а затем с помощью метода **join()** преобразовать список обратно в строку. Пример:



**7.7 Изменение регистра символов**

Для изменения регистра символов предназначены следующие методы:

* **upper()** - заменяет все символы строки соответствующими прописными буквами:



* **lower()** - заменяет все символы строки соответствующими строчными буквами:



* **swapcase()** - заменяет все строчные символы соответствующими прописными буквами, а все прописные символы -строчными:



* **capitalize()** -делает первую букву прописной:



* **title()** - делает первую букву каждого слова прописной:



**7.8 Функции для работы ·с ·символами**

Для работы с отдельными символами предназначены следующие функции:

* **chr (<Код символа> )** - возвращает символ по указанному коду:



* **ord (<Символ>)** - возвращает код указанного символа:



**7.9 Поиск и замена в строке**

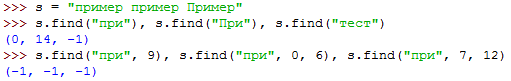
Для поиска и замены в строке используются следующие методы:

* **find()** - ищет подстроку в строке. Возвращает номер позиции, с которой начинается вхождение подстроки в строку. Если подстрока в строку не входит, то возвращается значение -1. Метод зависит от регистра символов. Формат метода:

**<Строка>.find(<Подстрока>[, <Начало>[, <Конец>]])**

Если начальная позиция не указана, то поиск будет осуществляться с начала строки.

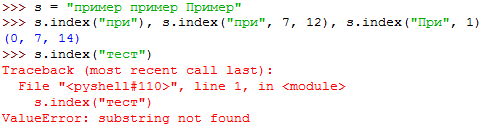
Если параметры <**Начало**> и <**Конец**> указаны, то производится операция извлечения среза **<Строка>[<Начало>:<Конец>]** и поиск подстроки будет выполняться в этом фрагменте. Пример:



* **index()** - метод аналогичен методу **find()**, но если подстрока в строку не входит, то возбуждается исключение **ValueError**. Формат метода:

**<Строка>.indех(<Подстрока>[, <Начало>[, <Конец>]])**

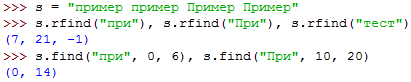
Пример:



* **rfind()** - ищет подстроку в строке. Возвращает позицию последнего вхождения подстроки в строку. Если подстрока в строку не входит, то возвращается значение -1. Метод зависит от регистра символов. Формат метода:

**<Строка>.rfind(<Подстрока>[, <Начало>[, <Конец>]])**

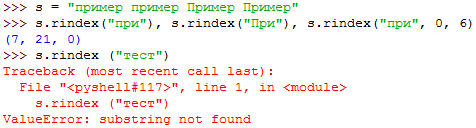
Если начальная позиция не указана, то поиск будет производиться с начала строки. Если параметры <**Начало**> и <**Конец**> указаны, то выполняется операция извлечения среза, и поиск подстроки будет производиться ·в этом фрагменте. Пример:



* **rindex()** - метод аналогичен методу **rfind()**, но если подстрока в строку не входит, то возбуждается исключение **Va1ueError**. Формат метода:

**<Строка>.rindех(<Подстрока>[, <Начало>[, <Конец>]])**

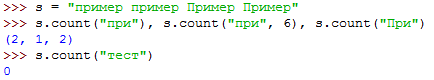
Пример:



* **count()** - возвращает число вхождений подстроки в строку. Если подстрока в строку не входит, то возвращается значение 0. Метод зависит от регистра символов. Формат метода:

**<Строка>.соunt(<Подстрока>[, <Начало>[, <Конец>]])**

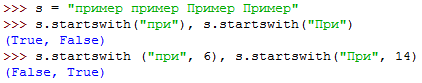
Пример:



* **startswith()** - проверяет, начинается ли строка с указанной подстроки. Если начинается, то возвращается значение **True**, в противном случае - **False**. Метод зависит от регистра символов. Формат метода:

**<Cтpoкa>.startswith(<Пoдcтpoкa>[, <Начало>[, <Конец>]])**

Если начальная позиция не указана, сравнение будет производиться с началом строки. Если параметры <**Начало**> и <**Конец**> указаны, то выполняется операция извлечения среза, и сравнение будет производиться с началом фрагмента. Пример:



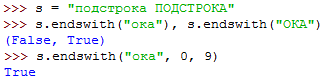
Начиная с версии **2.5**; параметр <**Подстрока**> может быть кортежем:



* **endswith()** - проверяет, заканчивается ли строка указанной подстрокой. Если заканчивается, то возвращается значение **True**, в противном случае - **Fa1se**. Метод зависит от регистра символов. Формат метода:

**<Строка>.еndswith(<Подстрока>[, <Начало>[, <Конец>]])**

Если начальная позиция не указана, то сравнение будет производиться с концом строки. Если параметры <**Начало**> и <**Конец**> указаны, то выполняется операция извлечения среза, и сравнение будет производиться с концом фрагмента. Пример:



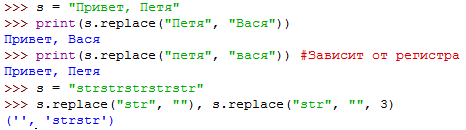
Начиная с версии **2.5**, параметр <**Подстрока**> может быть кортежем:



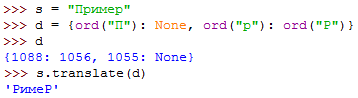
* **rep1ace()** - производит замену всех вхождений подстроки в строку на другую подстроку и возвращает результат в виде новой строки. Метод зависит от регистра символов. Формат метода:

**<Строка>.rер1асе(<Подстрока для замены, <Новая подстрока>[,<Максимальное количество замен>])**

Пример:



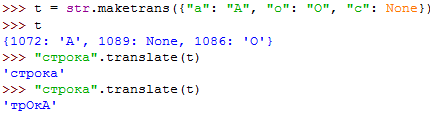
* **translate (<Таблица символов>)** - заменяет символы в соответствии с параметром <**Таблица символов**>. Параметр <**Таблица символов**> должен быть словарем, ·ключами которого являются **Unicode-кoды** заменяемых символов, а значениями - **Unicode-кoды** вставляемых символов. Если в качестве значения указать **None**, то символ будет удален. Для примера удалим букву "**П**", а также изменим регистр всех букв "**р**":



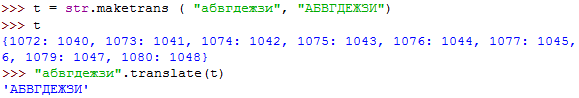
Упростить создание <**Таблицы символов**> позволяет статический метод **maketrans()**. Формат метода:

**str.maketrans(<X>[, <У>[, <Z>]])**

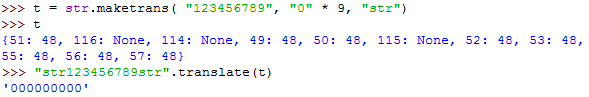
Если указан только первый параметр, то он должен быть словарем:



Если указаны два первых параметра, то они должны быть строками одинаковой длины. В результате будет создан словарь с ключами из строки <**Х**> и значениями, расположенными в той же позиции, из строки <**У**>. Изменим регистр некоторых символов:



В третьем параметре можно дополнительно указать строку из символов, которым будет сопоставлено значение **None**. После выполнения метода translate() эти символы будут удалены из строки. Заменим все цифры на 0, а некоторые буквы удалим из строки:



**7.10 Проверка типа содержимого строки**

Для проверки типа содержимого предназначены следующие методы:

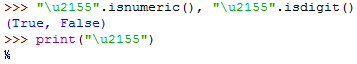
* **isdigit()** - возвращает **Тrue**, если строка содержит только цифры, в противном случае - **False**:



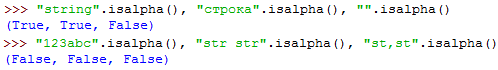
* **isdecimal()** - возвращает **True**, если строка содержит только десятичные символы, в противном случае- **False**. Пример:



* **isnumeric()** - возвращает **True**, если строка содержит только числовые символы, в противном случае - **False**. Обратите внимание на то, что к числовым символам относятся не только десятичные цифры, но символы римских чисел, дробные числа и др. Пример вывод символа "1/5":



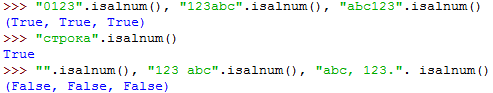
* **isalpha()** - возвращает **True**, если строка содержит только буквы, в противном случае - **False**. Если строка пустая, то возвращается значение **False**. Примеры:



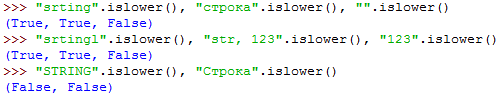
* **isspace()** -возвращает **True**, если строка содержит только пробельные символы, в противном случае - **False**:



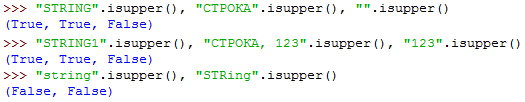
* **isalnum()** -возвращает **True**, если строка содержит только буквы и (или) цифры, в противном случае - **False**. Если строка пустая, то возвращается значение **False**. Примеры:



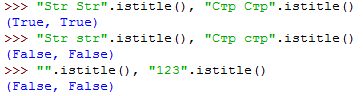
* **islower()** - возвращает **True**, если строка содержит буквы, и они все в нижнем регистре, в противном случае - **False**. Помимо букв строка может иметь другие символы, например цифры. Примеры:



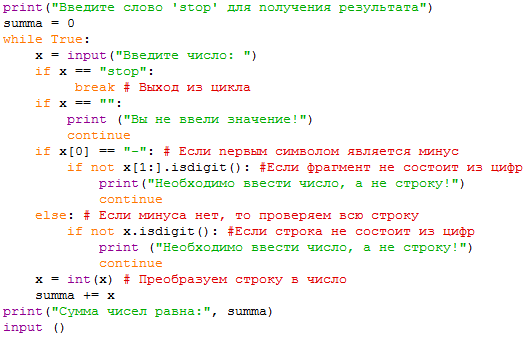
* **isupper()** - возвращает **True**, если строка содержит буквы, и они все в верхнем регистре, в противном случае - **False**. Помимо букв строка может иметь другие символы, например цифры. Примеры:



* **istitle()** - возвращает **True**, если строка содержит буквы, и первые буквы всех слов являются заглавными, в противном случае - **False**. Помимо букв строка может иметь другие символы, например цифры. Примеры:



Переделаем программу суммирования произвольного количества целых чисел, введенных пользователем, таким образом, чтобы при вводе строки вместо числа программа не завершалась с фатальной ошибкой.Кроме того, предусмотрим возможность ввода отрицательных целых чисел.



Процесс ввода значений и получения результата выглядит так:

Введите слово 'stop' для получения результата

