СИМВОЛЬНЫЕ СТРОКИ

Символьная строка – это последовательность символов, расположенных в памяти рядом (в соседних ячейках). Для работы с символами во многих языках программирования есть переменные специального типа: символы, символьные массивы и символьные строки (которые, в отличие от массивов, рассматриваются как цельные объекты). Основной тип данных для работы с символьными величинами в языке Python – это символьные строки (тип **string**).

Для того, чтобы записать в строку значение, используют оператор присваивания

```
s = "Вася пошёл гулять"
```

Строка заключается в кавычки или в одиночные апострофы. Если строка ограничена кавычками, внутри неё могут быть апострофы, и наоборот.

Для ввода строки с клавиатуры применяют функцию **input**:

```
s = input( "Введите имя: " )
```

Длина строки определяется с помощью функции **len** (от англ. *length* – длина). В этом примере в переменную n записывается длина строки **s**:

```
n = len(s)
```

Для того, чтобы выделить отдельный символ строки, к нему нужно обращаться так же, как к элементам массива (списка): в квадратных скобках записывают номер символа. Например, так можно вывести на экран символ строки s с индексом 5 (длина строки в этом случае должна быть не менее 6 символов):

```
print ( s[5] )
```

Отрицательный индекс говорит о том, что отсчёт ведётся с конца строки. Например, **s[-1]** означает то же самое, что **s[len(s)-1]**, то есть последний символ строки.

В отличие от большинства современных языков программирования, в Python нельзя изменить символьную строку, поскольку строка – это неизменяемый объект. Поэтому оператор присваивания **s[5] = "a"** не сработает – будет выдано сообщение об ошибке.

Тем не менее, можно составить из символов существующей строки новую строку, и при этом внести нужные изменения. Приведём полную программу, которая вводит строку с клавиатуры, заменяет в ней все буквы **"а"** на буквы **"б"** и выводит полученную строку на экран.

```
s = input( "Введите строку:" )
s1 = ""
for c in s:
  if c == "а": c = "б"
  s1 = s1 + c
print ( s1 )
```

Здесь в цикле **for c in s** перебираются все символы, входящие в строку **s**. Каждый их них на очередном шаге, записывается в переменную **c**. Затем мы проверяем значение этой переменной: если оно совпадает с буквой «а», то заменяем его на букву «б», и прицепляем в конец новой строки **s1** с помощью оператора сложения.

Нужно отметить, что показанный здесь способ (многократное «сложение» строк) работает очень медленно. В практических задачах, где требуется замена символов, лучше использовать стандартную функцию **replace**, о которой пойдет речь дальше.

Операции со строками

Как мы уже видели, оператор '+' используется для объединения (сцепления) строк, эта операция иногда называется конкатенация. Например:

```
s1 = "Привет"
s2 = "Вася"
s  = s1 + ",  " + s2 + "!"
```

В результате выполнения приведённой программы в строку s будет записано значение "Привет, Вася!".

Для того, чтобы выделить часть строки (*подстроку*), в языке Python применяется операция получения среза (англ. *slicing*), например **s[3:8]** означает символы строки s с 3-го по 7-й (то есть до

8-го, не включая его). Следующий фрагмент копирует в строку **s1** символы строки s с 3-го по 7-й (всего 5 символов):

```
s = "0123456789"
s1 = s[3:8]
```

В строку s1 будет записано значение "34567".

Для удаления части строки нужно составить новую строку, объединив части исходной строки до и после удаляемого участка:

```
s = "0123456789"
s1 = s[:3] + s[9:]
```

Срез s[:3] означает «от начала строки до символа s[3], не включая его», а запись s[9:] –«все символы, начиная с s[9] до конца строки». Таким образом, в переменной s1 остаётся значение "0129".

С помощью срезов можно вставить новый фрагмент внутрь строки:

```
s = "0123456789"
s1 = s[:3] + "ABC" + s[3:]
```

Переменная **s** получит значение "**012ABC3456789**".

Если заданы отрицательные индексы, к ним добавляется длина строки **N**. Таким образом, индекс «–1» означает то же самое, что **N-1**. При выполнении команд

```
s = "0123456789"
s1 = s[:-1]          # "012345678"
s2 = s[-6:-2]        # "4567"
```

мы получим **s1 = "012345678"** (строка s кроме последнего символа) и **s2 = "4567".**

Срезы позволяют выполнить реверс строки (развернуть её наоборот):

```
s1 = s[::-1]
```

Так как начальный и конечный индексы элементов строки не указаны, задействована вся строка. Число «–1» означает шаг изменения индекса и говорит о том, что все символы перебираются в обратном порядке.

В Python существует множество встроенных методов для работы с символьными строками. Например, методы **upper** и **lower** позволяют перевести строку соответственно в верхний и нижний регистр:

```
s = "aAbBcC"
s1 = s.upper()  # "AABBCC"
s2 = s.lower()  # "aabbcc"
```

а метод isdigit проверяет, все ли символы строки – цифры, и возвращает логическое значение:

```
s = "abc"
print ( s.isdigit() )   # False
s1 = "123"
print ( s1.isdigit() )  # True
```

О других встроенных функциях вы можете узнать в литературе или в Интернете.

Поиск в строках

В Python существует функция для поиска подстроки (и отдельного символа) в символьной строке. Эта функция называется **find**, она определена для символьных строк и вызывается как метод, с помощью точечной записи. В скобках нужно указать образец для поиска:

```
s = "Здесь был Вася."
n = s.find ( "с" )        # n = 3
if n >= 0:
  print ( "Номер символа", n )
else:
  print ( "Символ не найден." )
```

Метод **find** возвращает целое число – номер символа, с которого начинается образец (буква "с") в строке s. Если в строке несколько образцов, функция находит первый из них. Если образец не найден, функция возвращает «–1». В рассмотренном примерев переменную n будет записано число 3.

Аналогичный метод **rfind** (от англ. *reverse find* – искать в обратную сторону) ищет последнее вхождение образца в строке. Для той же строки s, что и в предыдущем примере, метод **rfind** вернёт 12 (индекс последней буквы «с»):

```
s = "Здесь был Вася."
n = s.rfind ( "с" )       # n = 12
```

Пример обработки строки

Предположим, что с клавиатуры вводится строка, содержащая имя, отчество и фамилию человека, например:

Василий Алибабаевич Хрюндиков

Каждые два слова разделены одним пробелом, в начале строки пробелов нет. В результате обработки должна получиться новая строка, содержащая фамилию и инициалы:

Хрюндиков В.А.

Возможный алгоритм решения этой задачи может быть на псевдокоде записан так:

```
ввести строку s
найти в строке s первый пробел
имя = всё, что слева от первого пробела
удалить из строки s имя с пробелом
найти в строке s первый пробел
отчество = всё, что слева от первого пробела
удалить из строки s отчество с пробелом   # осталась фамилия
s = s + " " + имя[0] + "." + отчество[0] + "."
```

Мы последовательно выделяем из строки три элемента: имя, отчество и фамилию, используя тот факт, что они разделены одиночными пробелами. После того, как имя сохранено в отдельной переменной, в строке **s** остались только отчество и фамилия. После «изъятия» отчества остается только фамилия. Теперь нужно собрать строку-результат из частей: «сцепить» фамилию и первые буквы имени и отчества, поставив пробелы и точки между ними.

Для выполнения всех операций будем срезы и метод **find**. Приведем сразу полную программу:

```
print ( "Введите имя, отчество и фамилию:" )
s = input()
n = s.find ( " " )
name = s[:n]          # вырезать имя
s = s[n+1:]
n = s.find ( " " )
name2 = s[:n]         # вырезать отчество
s = s[n+1:]           # осталась фамилия
s = s + " " + name[0] + "." + name2[0] + "."
print ( s )
```

Используя встроенные функции языка Python, эту задачу можно решить намного проще. Прежде всего, нужно разбить введённую строку на отдельные слова, которые разделены пробелами. Для этого используется метод **split** (от англ. *split* – расщепить) который возвращает список слов, полученных при разбиении строки:

```
fio = s.split()
```

Если входная строка правильная (соответствует формату, описанному в условии), то в этом списке будут три элемента: **fio[0]** – имя, **fio[1]** – отчество и **fio[2]** – фамилия. Теперь остаётся только собрать строку-результат в нужном виде:

```
s = fio[2] + " " + fio[0][0] + "." + fio[1][0] + "."
```

Запись **fio[0][0]** обозначает «0-й символ из 0-го элемента списка fio», то есть, первая буква имени. Аналогично **fio[1][0]** – это первая буква отчества. Вот полная программа:

```
print ( "Введите имя, отчество и фамилию:" )
s = input()
fio = s.split()
s = fio[2] + " " + fio[0][0] + "." + fio[1][0] + "."
print ( s )
```

Преобразования число-строка

В практических задачах часто нужно преобразовать число, записанное в виде цепочки символов, в числовое значение, и наоборот. Для этого в языке Python есть стандартные функции:

- **int** – переводит строку в целое число;
- **float** – переводит строку в вещественное число;
- **str** – переводит целое или вещественное число в строку.

Приведём пример преобразования строк в числовые значения:

```
s = "123"
N = int ( s )        # N = 123
s = "123.456"
X = float ( s )      # X = 123.456
```

Если строку не удалось преобразовать в число (например, если в ней содержатся буквы), возникает ошибка и выполнение программы завершается.

Теперь покажем примеры обратного преобразования:

```
N = 123
s = str ( N )    # s = "123"
X = 123.456
s = str ( X )    # s = "123.456"
```

Эти операции всегда выполняются успешно (ошибка произойти не может).

Функция **str** использует правила форматирования, установленные по умолчанию. При необходимости можно использовать собственное форматирование. Например, в строке

```
s = "{:5d}".format(N)
```

значение переменной N будет записано по формату **d** (целое число) в 5 позициях, то есть в начале строки будут стоять два пробела:

```
◦◦123
```

Для вещественных чисел можно использовать форматы **f** (с фиксированной точкой) и **e** (экспоненциальный формат, с плавающей точкой):

```
X = 123.456
s = "{:7.2f}".format(X)    # s = "◦123.46"
s = "{:10.2e}".format(X)   # s = "◦◦1.23e+02"
```

В первом случае формат «**7.2f**» обозначает «вывести в 7 позициях с 2 знаками в дробной части», во втором – формат «**10.2e**» обозначает «вывести научном формате в 10 позициях с 2 знаками в дробной части».

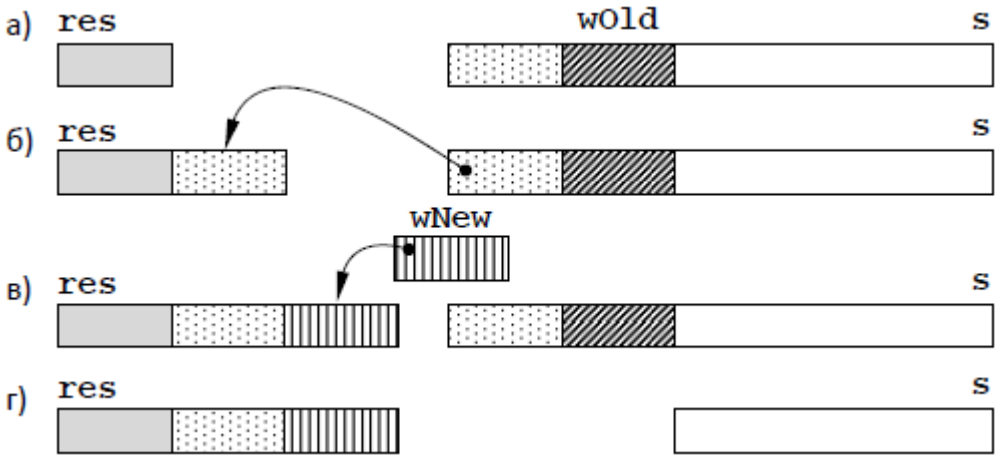
Строки в процедурах и функциях

Строки можно передавать в процедуры и функции как параметры, а также возвращать как результат функций. Построим процедуру, которая заменяет в строке **s** все вхождения слова образца **wOld** на слово-замену **wNew** (здесь **wOld** и **wNew** – это имена переменных, а выражение «слово **wOld**» означает «слово, записанное в переменную **wOld**»).

Сначала разработаем алгоритм решения задачи. В первую очередь в голову приходит такой псевдокод

Однако такой алгоритм работает неверно, если слово **wOld** входит в состав **wNew**, например, нужно заменить **"12"** на **"A12B"** (покажите самостоятельно, что это приведет к зацикливанию).

Чтобы избежать подобных проблем, попробуем накапливать результат в другой символьной строке **res**, удаляя из строки **s** уже обработанную часть. Предположим, что на некотором шаге в оставшейся части строки **s** обнаружено слово **wOld**(рис. а)



Теперь нужно выполнить следующие действия:

1) ту часть строки **s**, которая стоит слева от образца, «прицепить» в конец строки **res** (рис. б);
2) «прицепить» в конец строки **res** слово-замену **wNew** (рис. в);
3) удалить из строки **s** начальную часть, включая найденное слово-образец (рис. г).

Далее все эти операции (начиная с поиска слова **wOld** в строке **s**) выполняется заново до тех пор, пока строка **s** не станет пустой. Если очередное слово найти не удалось, вся оставшаяся строка **s** приписывается в конец строки-результата, и цикл заканчивается.

В начале работы алгоритмы в строку **res** записывается пустая строка **""**, не содержащая ни одного символа. В таблице приведен протокол работы алгоритма замены для строки**"12.12.12"**, в которой нужно заменить слово **"12"** на **"A12B"**

рабочая строка s	результат res
"12.12.12"	""
".12.12"	"A12B"
".12"	"A12B.A12B"
""	"A12B.A12B.A12B"

Теперь можно написать функцию на языке Python. Её параметры – это исходная строка s, строка-образец wOld и строка-замена wNew:

```
def replaceAll ( s, wOld, wNew ):
  lenOld = len(wOld)
  res = ""
  while len(s) > 0:
    p = s.find ( wOld )
    if p < 0:
      return res + s
    if p > 0:
      res = res + s[:p]
    res = res + wNew
    if p+lenOld >= len(s):
      s = ""
    else:
      s = s[p+lenOld:]
  return res
```

Переменная **p** – это номер первого символа первого найденного слова-образца **wOld**, а в переменной **lenOld** записана длина этого слова. Если после поиска слова значение **p** меньше нуля (образец не найден), происходит выход из цикла:

```
if p < 0: res = res + s; return
```

Если **p >** 0, то слева от образца есть какие-то символы, и их нужно «прицепить» к строке **res**:

```
if p > 0: res = res + s[:p]
```

Условие **p+lenOld >= len(s)** означает «образец стоит в самом конце слова», при этом остаток строки **s** – пустая строка. В конце программы результат записывается на место исходной строки **s**.

Приведем пример использования этой функции:

```
s = "12.12.12"
s = replaceAll ( s, "12", "A12B" )
print ( s )
```

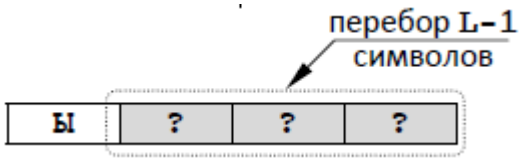
Так как операция замены одной подстроки на другую используется очень часто, в языке Python есть встроенная функция, которая выполняет эту операцию. Она оформлена как метод для переменных типа «строка» (**str**) и вызывается с помощью точечной записи:

```
s = "12.12.12"
s = s.replace( "12", "A12B" )
print ( s )
```

Рекурсивный перебор

В алфавите языке племени «тумба-юмба» четыре буквы: «**Ы**», «**Ш**», «**Ч**» и «**О**». Нужно вывести на экран все слова, состоящие из **L** букв, которые можно построить из букв этого алфавита.

Это типичная задача на перебор вариантов, которую удобно свести к задаче меньшего размера. Будем определять буквы слова последовательно, одну за другой. Первая буква может быть любой из четырёх букв алфавита. Предположим, что сначала первой мы поставили букву **'Ы'**. Тогда для того, чтобы получить все варианты с первой буквой **'Ы'**, нужно перебрать все возможные комбинации букв на оставшихся **L-1** позициях:



Далее поочередно ставим на первое место все остальные буквы, повторяя процедуру:

```
def перебор L символов:
  w = "Ы"; перебор последних L-1 символов
  w = "Ш"; перебор последних L-1 символов
  w = "Ч"; перебор последних L-1 символов
  w = "О"; перебор последних L-1 символов
```

Здесь через w обозначена символьная строка, в которой собирается рабочее слово. Таким образом, задача для слов длины L свелась к 4-м задачам для слов длины L-1. Как вы знаете, такой прием называется рекурсией, а процедура – рекурсивной.

Когда рекурсия должна закончиться? Тогда, когда все символы расставлены, то есть длина строки w станет равна L. При этом нужно вывести получившееся слово на экран и выйти из процедуры.

Подсчитаем количество всех возможных слов длины L. Очевидно, что слов длины 1 всего 4. Добавляя ещё одну букву, получаем 4·4=16 комбинаций, для трех букв – 4·4·4=64 слова и т.д. Таким образом, слов из L букв может быть 4L.

Процедура для перебора слов может быть записана так:

```
def TumbaWords ( A, w, L ):
  if len(w) == L:
    print ( w )
    return
  for c in A:
    TumbaWords ( A, w+c, L )
```

В параметре **A** передаётся алфавит языка, параметр **w** – это уже построенная часть слова, а **L** – нужная длина слова. Когда длина построенного слова станет равна **L**, то есть будет получено слово требуемой длины, слово выводится на экран и происходит выход из процедуры (окончание

рекурсии). Если слово не достроено, в цикле перебираем все символы, входящие в алфавит, по очереди добавляем каждый из них в конец строки и вызываем процедуру рекурсивно.
В основной программе остаётся вызвать эту процедуру с нужными исходными данными

```
TumbaWords ( "ЫШЧО", "", 3 )
```

При таком вызове будут построены все трёхбуквенные слова, которые можно получить с помощью алфавита «ЫШЧО».

Сравнение и сортировка строк

Строки, как и числа, можно сравнивать. Для строк, состоящих из одних букв (русских или латинских), результат сравнения очевиден: меньше будет та строка, которая идет раньше в алфавитном порядке. Например, слово «паровоз» будет «меньше», чем слово «пароход»: они отличаются в пятой букве и «в» < «х». Более короткое слово, которое совпадает с началом более длинного, тоже будет стоять раньше в алфавитном списке, поэтому «пар» < «парк».

Но как откуда компьютер «знает», что такое «алфавитный порядок»? И как сравнивать сло-ва, в которых есть и строчные и заглавные буквы, а также цифры и другие символы. Что больше,«ПАР», «Пар» или «пар»?

Оказывается, при сравнении строк используются коды символов. Тогда получается, что

«ПАР» < «Пар» < «пар».

Возьмем пару «ПАР» и «Пар». Первый символ в обоих словах одинаков, а второй отличается – впервом слове буква заглавная, а во втором – такая же, но строчная. В таблице символов заглавныебуквы стоят раньше строчных, и поэтому имеют меньшие коды. Поэтому «А» < «а», «П» < «а» и«ПАР» < «Пар» < «пар».

А как же с другими символами (цифрами, латинскими буквами)? Цифры стоят в кодовой таблице по порядку, причём раньше, чем латинские буквы; латинские буквы – раньше, чем русские; заглавные буквы (русские и латинские) – раньше, чем соответствующие строчные. Поэтому

«5STEAM» < «STEAM» < «Steam» < «steam» < «ПАР» < «Пар» < «пар».

Сравнение строк используется, например, при сортировке. Рассмотрим такую задачу: ввести с клавиатуры несколько слов (например, фамилий) и вывести их на экран в алфавитном порядке.

Для решения этой задачи с помощью языка Python удобно записать строки в массив (список)и затем отсортировать с помощью метода **sort**:

```
aS = []
print ( "Введите строки для сортировки:" )
while True:
  s1 = input()
  if s1 == "": break
  aS.append ( s1 )
aS.sort()
print ( aS )
```

Строки заносятся в список **aS**. Сначала этот список пустой, затем в цикле мы вводим очередную строку с клавиатуры и записываем её в переменную **s1**. Ввод заканчивается, когда введена пустая строка, то есть вместо ввода строки пользователь нажал клавишу *Enter*. В этом случае сработает условный оператор и выполнится оператор **break**, прерывающий цикл.