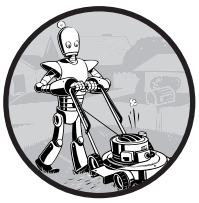
6

МАНИПУЛИРОВАНИЕ СТРОКАМИ



Текст — одна из наиболее распространенных форм данных, которые будут обрабатываться вашими программами. Вы уже знаете, как конкатенировать два строковых значения с помощью оператора +, но ваши возможности гораздо шире. Python позволяет извлекать части строк из строковых значений, добавлять и удалять пробелы, преобразовывать буквы из нижнего регистра в верхний и обратно, а также форматировать строки по своему желанию. Вы даже можете написать код на Python для доступа к буферу обмена, используя его для копирования и вставки текста.

В этой главе мы поработаем над двумя программными проектами: простым менеджером паролей и программой для автоматизации рутинной работы по форматированию текста.

Работа со строками

Приступим к рассмотрению различных способов записи строк, а также их вывода на экран и получения доступа к ним с помощью кода на языке Python.

Строковые литералы

Ввод строк в коде Python не составляет труда: они начинаются и заканчиваются символами апострофа (одинарными кавычками). Но как быть, если кавычки требуются в самой строке? Ввод строки в виде `'That is Alice's cat.'` не сработает, поскольку Python интерпретирует вторую кавычку после слова `Alice` как конец строки и сочтет остальную часть текста (`s cat.'`) недопустимым кодом. К счастью, существует несколько способов ввода строк.

Кавычки

Начало и конец строки можно обозначать не только парой апострофов, но и парой кавычек. Одно из преимуществ использования кавычек — в том, что они позволяют включать символ апострофа в состав строки. Введите в интерактивной оболочке следующий код:

```
>>> spam = "That is Alice's cat."
```

Поскольку строка начинается с кавычки, Python в состоянии идентифицировать апостроф как часть строки. Если же в составе строки требуются как апострофы, так и кавычки, то необходимо прибегать к экранированию символов.

Экранированные символы

Техника экранирования позволяет использовать символы, вставка которых в строку иными способами невозможна. *Экранированный символ* включает символ обратной косой черты (\), за которым следует сам символ, добавляемый в строку. (Несмотря на то что экранированный символ состоит из двух символов, его рассматривают как одиночный.) Вот так, например, выглядит экранированный апостроф: `\'`. Его можно использовать даже в строке, которая начинается и заканчивается апострофом. Чтобы увидеть, как работают экранированные символы, введите в интерактивной оболочке следующий код:

```
>>> spam = 'Say hi to Bob\'s mother.'
```

Поскольку в слове `Bob\'s` апострофу предшествует символ обратной косой черты, Python знает, что в данном случае апостроф не служит маркером конца строки. Экранированные символы `\'` и `\"` позволяют включать в строку соответственно апострофы и кавычки.

Экранированные символы, которые можно использовать, приведены в табл. 6.1.

Таблица 6.1. Экранированные символы

Экранированный символ	Отображаемый символ
`\'`	Апостроф
`\"`	Кавычка
`\t`	Символ табуляции
`\n`	Символ новой строки (разрыва строки)
`\\`	Символ обратной косой черты

Введите в интерактивной оболочке следующую команду.

```
>>> print("Hello there!\nHow are you?\nI\'m doing fine.")
Hello there!
How are you?
I'm doing fine.
```

"Сырые" строки

Поместив символ `r` перед начальной кавычкой, вы обозначаете строку как "сырую". *Сырая* (т.е. необработанная) строка полностью игнорирует механизм экранирования и выводит все символы обратной косой черты, которые встречаются в строке, как обычные символы. Введите, например, в интерактивной оболочке следующую команду.

```
>>> print(r'That is Carol\'s cat.')
That is Carol\'s cat.
```

Поскольку это "сырая" строка, Python считает все символы обратной косой черты ее частью, а не началом экранированного символа. "Сырые" строки полезны в тех случаях, когда вы вводите строковые значения с многочисленными символами обратной косой черты, как это бывает при использовании регулярных выражений, описанных в следующей главе.

Многострочные блоки

Несмотря на то что вы всегда можете включить в строку символ новой строки с помощью экранированного символа `\n`, во многих случаях гораздо проще использовать многострочные блоки. В Python многострочные блоки представляют собой группу строк, заключенных в тройные апострофы или кавычки. Любые апострофы, кавычки или символы новой строки в блоке, ограниченном такими "тройными кавычками", считаются частью строки. Правила Python, регламентирующие форматирование блоков кода с помощью отступов, в отношении многострочных блоков не действуют.

Откройте файловый редактор и введите следующий код.

```
print('''Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary, and
↳ extortion.

Sincerely,
Bob''')
```

Сохраните эту программу в файле *catnapping.py* и выполните ее. Вы должны получить следующий вывод.

```
Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary, and
extortion.

Sincerely,
Bob
```

Обратите внимание на то, что для символа апострофа в слове `Eve's` экранирование не понадобилось. В “сырых” строках экранирование апострофов и кавычек необязательно. Для вывода текста в таком же виде без использования многострочного блока потребуется следующий вызов функции `print()`.

```
print('Dear Alice,\n\nEve\'s cat has been arrested for catnapping,
↳ cat burglary, and extortion.\n\nSincerely,\nBob')
```

Многострочные комментарии

В то время как символом “решетка” (`#`) обозначается однострочный комментарий, длина которого ограничивается концом строки, многострочные блоки часто используют в качестве многострочных комментариев, охватывающих произвольное количество строк. Приведенный ниже текст абсолютно корректен в Python.

```
"""This is a test Python program.
Written by Al Sweigart al@inventwithpython.com

This program was designed for Python 3, not Python 2.
"""

def spam():
    """This is a multiline comment to help
    explain what the spam() function does."""
    print('Hello!')
```

Индексирование строк и извлечение срезов

В случае строк операции индексирования и извлечения срезов выполняются точно так же, как и в случае списков. Можете рассматривать строку `'Hello world!'` как список, в котором каждый символ строки является элементом списка и имеет соответствующий индекс.

```
'   H   e   l   l   o       w   o   r   l   d   !   ,
    0   1   2   3   4   5   6   7   8   9   10  11
```

Пробелы и восклицательный знак входят в число символов, поэтому фраза `'Hello world!'` содержит 12 символов, от символа `H` с индексом `0` до символа `!` с индексом `11`.

Введите в интерактивной оболочке следующие команды.

```
>>> spam = 'Hello world!'
  spam[0]
'H'
>>> spam[4]
'o'
>>> spam[-1]
'!'
>>> spam[0:5]
'Hello'
>>> spam[:5]
'Hello'
>>> spam[6:]
'world!'
```

Указав индекс, вы получаете символ, находящийся в соответствующей позиции в строке. В случае указания диапазона индексов, т.е. извлечения среза, элемент с начальным индексом включается в срез, тогда как элемент с конечным индексом не включается. Поэтому, если `spam` — это строка `'Hello world!'`, то срез `spam[0:5]` содержит строку `'Hello'`. Подстрока, получаемая с помощью среза `spam[0:5]`, будет включать в себя все символы от `spam[0]` до `spam[4]`, тогда как символ пробела, имеющий индекс `5`, в нее не войдет.

Имейте в виду, что извлечение части строки посредством среза не сопровождается изменением исходной строки. Вы можете сохранить срез, извлеченный из одной переменной, в другой переменной. Введите в интерактивной оболочке следующие команды.

```
>>> spam = 'Hello world!'
>>> fizz = spam[0:5]
>>> fizz
'Hello'
```

Извлекая срез и сохраняя результирующую подстроку в другой переменной, вы получаете быстрый и простой доступ как ко всей строке, так и к ее подстроке.

Использование операторов `in` и `not in` со строками

Операторы `in` и `not in` применяются к строкам точно так же, как и к спискам. Результатом вычисления выражения в виде двух строк, соединенных любым из этих операторов, является булево значение `True` или `False`. Введите в интерактивной оболочке следующие команды.

```
>>> 'Hello' in 'Hello World'
True
>>> 'Hello' in 'Hello'
True
>>> 'HELLO' in 'Hello World'
False
>>> '' in 'spam'
True
>>> 'cats' not in 'cats and dogs'
False
```

Эти выражения проверяют, содержится ли первая строка (в строгом соответствии с регистром набора символов) во второй строке.

Полезные методы для работы со строками

Некоторые методы анализируют строки или создают преобразованные строковые значения. В этом разделе описаны те из них, которые вы будете использовать наиболее часто.

Методы `upper()`, `lower()`, `isupper()` и `islower()`

Методы `upper()` и `lower()` возвращают новую строку, в которой все буквы исходной строки преобразованы соответственно в верхний или нижний регистр. Небуквенные символы не испытывают никаких изменений. Введите в интерактивной оболочке следующий код.

```
>>> spam = 'Hello world!'
>>> spam = spam.upper()
>>> spam
'HELLO WORLD!'
>>> spam = spam.lower()
>>> spam
'hello world!'
```

Имейте в виду, что эти методы возвращают новые строковые значения, не изменяя саму исходную строку. Чтобы изменить исходную строку, необходимо вызвать для нее метод `upper()` или `lower()` и присвоить результирующее строковое значение той же переменной, в которой хранилась

исходная строка. Именно поэтому для того, чтобы изменить строку, хранящуюся в переменной `spam`, следует выполнить операцию присваивания `spam = spam.upper()`, а не просто вызвать функцию `spam.upper()`. (В качестве аналогии можно привести переменную `eggs`, содержащую значение `10`. Выражение `eggs + 3` не изменит значения `eggs`, но это можно сделать с помощью инструкции `eggs = eggs + 3`.)

Методы `upper()` и `lower()` удобно применять в тех случаях, когда при сравнении строк не должен учитываться регистр букв. Строки `'great'` и `'GREat'` — это разные строки. Однако в приведенной ниже небольшой программе не имеет значения, в какой форме пользователь введет слово, `Great`, `GREAT` или `grEAT`, поскольку строка предварительно преобразуется в верхний регистр.

```
print('How are you?')
feeling = input()
if feeling.lower() == 'great':
    print('I feel great too.')
else:
    print('I hope the rest of your day is good.')
```

Когда вы запустите эту программу, на экране отобразится вопрос, и даже если вы ответите на него, введя слово `GREat`, все равно будет выведена строка `I feel great too`. Добавляя в свою программу код, нивелирующий различия в написании слов и игнорирующий такие ошибки, как неправильное использование строчных и прописных букв, вы упростите работу с ней и уменьшите вероятность ее аварийного завершения из-за ошибок, допущенных пользователем при вводе текста.

```
How are you?
GREat
I feel great too.
```

Методы `isupper()` и `islower()` возвращают булево значение `True`, если в строке имеется хотя бы одна буква и все буквы относятся соответственно к верхнему или нижнему регистру. В противном случае метод возвращает значение `False`. Введите в интерактивной оболочке следующие команды и обратите внимание на возвращаемые методами значения.

```
>>> spam = 'Hello world!'
>>> spam.islower()
False
>>> spam.isupper()
False
>>> 'HELLO'.isupper()
```

```
True
>>> 'abc12345'.islower()
True
>>> '12345'.islower()
False
>>> '12345'.isupper()
False
```

Поскольку методы `upper()` и `lower()` сами возвращают строки, для этих строк, в свою очередь, также можно вызывать строковые методы. Выражения, с помощью которых это делается, выглядят как цепочки вызовов методов. Введите в интерактивной оболочке следующие команды.

```
>>> 'Hello'.upper()
'HELLO'
>>> 'Hello'.upper().lower()
'hello'
>>> 'Hello'.upper().lower().upper()
'HELLO'
>>> 'HELLO'.lower()
'hello'
>>> 'HELLO'.lower().islower()
True
```

Строковые методы `isX()`

Наряду с функциями `islower()` и `isupper()` существует ряд методов для работы со строками, имена которых начинаются со слова `is`. Методы этой категории возвращают булево значение, описывающее природу строки, для которой они вызываются. Ниже приведен перечень часто используемых строковых методов типа `isX()`.

- `isalpha()` возвращает значение `True`, если строка состоит только из букв и не является пустой.
- `isalnum()` возвращает значение `True`, если строка состоит только из буквенно-цифровых символов и не является пустой.
- `isdecimal()` возвращает значение `True`, если строка состоит только из цифровых символов и не является пустой.
- `isspace()` возвращает значение `True`, если строка состоит только из символов пробела, табуляции, новой строки и не является пустой.
- `istitle()` возвращает значение `True`, если строка состоит только из слов, которые начинаются с буквы в верхнем регистре, за которой следуют только буквы в нижнем регистре.

Введите в интерактивной оболочке следующие команды.

```
>>> 'hello'.isalpha()
True
>>> 'hello123'.isalpha()
False
>>> 'hello123'.isalnum()
True
>>> 'hello'.isalnum()
True
>>> '123'.isdecimal()
True
>>> ' '.isspace()
True
>>> 'This Is Title Case'.istitle()
True
>>> 'This Is Title Case 123'.istitle()
True
>>> 'This Is not Title Case'.istitle()
False
>>> 'This Is NOT Title Case Either'.istitle()
False
```

Методы `isX()` удобно применять для проверки допустимости введенных пользователем данных. Например, приведенная ниже программа повторяет запрос ввода пользователем своего возраста и пароля до тех пор, пока эти данные не будут предоставлены в корректном формате. Откройте новое окно в файловом редакторе, введите в нем следующий код программы и сохраните его в файле *validateInput.py*.

```
while True:
    print('Enter your age:')
    age = input()
    if age.isdecimal():
        break
    print('Please enter a number for your age.')

while True:
    print('Select a new password (letters and numbers only):')
    password = input()
    if password.isalnum():
        break
    print('Passwords can only have letters and numbers.')
```

В первом цикле `while` программа запрашивает ввод возраста пользователя и сохраняет введенное значение. Если пользователь ввел возраст в виде допустимого значения (десятичного числа), первый цикл прерывается и управление передается второму циклу `while`, в котором запрашивается пароль. В противном случае программа информирует пользователя о том, что должно быть введено число, и предлагает повторно указать возраст.

Во втором цикле `while` запрашивается и сохраняется пароль. Если пользователь ввел буквенно-цифровое значение, выполняется выход из цикла. В противном случае программа информирует пользователя о том, что допускаются только пароли, состоящие из букв и цифр, и предлагает ему повторить ввод.

Запустив программу, вы должны получить примерно такой вывод.

```
Enter your age:
forty two
Please enter a number for your age.
Enter your age:
42
Select a new password (letters and numbers only):
secr3t!
Passwords can only have letters and numbers.
Select a new password (letters and numbers only):
secr3t
```

Вызывая методы `isdecimal()` и `isalnum()` для переменных, мы можем выяснить, являются ли введенные пользователем значения цифровыми или буквенно-цифровыми. В данном случае эти проверки позволили отвергнуть ввод пользователем строки `forty two` при указании возраста, но принять ввод числа `42`, а также отвергнуть ввод значения `secr3t!` в качестве пароля, но принять ввод значения `secr3t`.

Методы `startswith()` и `endswith()`

Методы `startswith()` и `endswith()` возвращают значение `True`, если строка, для которой они вызываются, соответственно начинается или заканчивается строкой, переданной методу; в противном случае они возвращают значение `False`. Введите в интерактивной оболочке следующие команды.

```
>>> 'Hello world!'.startswith('Hello')
True
>>> 'Hello world!'.endswith('world!')
True
>>> 'abc123'.startswith('abcdef')
False
>>> 'abc123'.endswith('12')
False
>>> 'Hello world!'.startswith('Hello world!')
True
>>> 'Hello world!'.endswith('Hello world!')
True
```

Эти методы — полезная альтернатива оператору сравнения ==, если сравнение с другой строкой требуется выполнить не для всей исходной строки, а только для первой или последней ее части.

Строковые методы `join()` и `split()`

Метод `join()` удобно использовать в тех случаях, когда ряд строк, представленных в виде списка, необходимо объединить в одно строковое значение. Метод `join()` вызывается для некоторой строки, принимает список строк в качестве аргумента и возвращает строку. Возвращаемая строка представляет собой результат конкатенации всех строк, переданных в виде списка. Введите в интерактивной оболочке следующие команды.

```
>>> ', '.join(['cats', 'rats', 'bats'])
'cats, rats, bats'
>>> ' '.join(['My', 'name', 'is', 'Simon'])
'My name is Simon'
>>> 'ABC'.join(['My', 'name', 'is', 'Simon'])
'MyABCnameABCisABCSimon'
```

Обратите внимание на то, что строка, для которой вызывается метод `join()`, вставляется между переданными посредством аргумента строками. Например, если выполнить вызов `join(['cats', 'rats', 'bats'])` для строки `', '`, то будет возвращена строка `'cats, rats, bats'`.

Не забывайте о том, что метод `join()` вызывается для строкового значения и в качестве аргумента принимает список. (Вы можете вызвать метод как-то иначе, сами того не осознавая.) Метод `split()` делает совершенно противоположное: он вызывается для строкового значения и возвращает список строк. Введите в интерактивной оболочке следующую команду.

```
>>> 'My name is Simon'.split()
['My', 'name', 'is', 'Simon']
```

По умолчанию строка `'My name is Simon'` разбивается на отдельные строки в тех местах, где встречаются пробельные символы: пробел, символ табуляции, символ новой строки. Эти пробельные символы не включаются в строки, возвращаемые в виде списка. Вы можете указать другую строку-разделитель, передав ее методу `split()`. Например, введите в интерактивной оболочке следующие команды.

```
>>> 'MyABCnameABCisABCSimon'.split('ABC')
['My', 'name', 'is', 'Simon']
>>> 'My name is Simon'.split('m')
['My na', 'e is Si', 'on']
```

Обычный способ применения метода split() — разбиение многострочного блока по символам новой строки. Введите в интерактивной оболочке следующие команды.

```
>>> spam = '''Dear Alice,
How have you been? I am fine.
There is a container in the fridge
that is labeled "Milk Experiment".

Please do not drink it.
Sincerely,
Bob'''
>>> spam.split('\n')
['Dear Alice,', 'How have you been? I am fine.', 'There is a
container in the fridge', 'that is labeled "Milk Experiment".',
'', 'Please do not drink it.', 'Sincerely,', 'Bob']
```

Передача методу split() строки '\n' в качестве аргумента позволяет выполнить разбивку многострочного блока, сохраненного в переменной spam, в позициях символов новой строки, и возвратить список, каждый элемент которого соответствует одной строке текста.

Выравнивание текста с помощью методов rjust(), ljust() и center()

Строковые методы rjust() и ljust() возвращают версию строки, для которой они вызываются, выровненную за счет вставки пробелов. В обоих методах первый аргумент — целое число, определяющее длину выровненной строки. Введите в интерактивной оболочке следующие команды.

```
>>> 'Hello'.rjust(10)
'  Hello'
>>> 'Hello'.rjust(20)
'              Hello'
>>> 'Hello World'.rjust(20)
'         Hello World'
>>> 'Hello'.ljust(10)
'Hello  '
```

Выражение 'Hello'.rjust(10) говорит о том, что мы хотим выровнять строку 'Hello' вправо так, чтобы полная длина строки составляла 10. В слове 'Hello' насчитывается пять символов, поэтому слева от него будут добавлены пять пробелов, в результате чего полная длина строки составит 10 символов, причем слово 'Hello' будет выровнено вправо.

Необязательный второй аргумент в обоих методах указывает на символ-заполнитель, отличный от пробела. Введите в интерактивной оболочке следующие команды.

```
>>> 'Hello'.rjust(20, '*')
'***************Hello'
>>> 'Hello'.ljust(20, '-')
'Hello---------------'
```

Метод `center()` работает аналогично методам `ljust()` и `rjust()`, но центрирует его, а не выравнивает по правому или левому краю. Введите в интерактивной оболочке следующие команды.

```
>>> 'Hello'.center(20)
' Hello '
>>> 'Hello'.center(20, '=')
'=======Hello========'
```

Эти методы особенно полезны в ситуациях, когда необходимо вывести табулированные значения с подходящей разрядкой. Откройте новое окно в файловом редакторе, введите в него следующий код и сохраните его в файле *picnicTable.py*.

```
def printPicnic(itemsDict, leftWidth, rightWidth):
    print('PICNIC ITEMS'.center(leftWidth + rightWidth, '-'))
    for k, v in itemsDict.items():
        print(k.ljust(leftWidth, '.') + str(v).rjust(rightWidth))
picnicItems = {'sandwiches': 4, 'apples': 12, 'cups': 4,
               'cookies': 8000}
printPicnic(picnicItems, 12, 5)
printPicnic(picnicItems, 20, 6)
```

В этой программе мы определяем метод `printPicnic()`, который получает данные в виде словаря и использует методы `center()`, `ljust()` и `rjust()` для отображения этих данных в аккуратном формате в виде выровненной страницы.

Функции `printPicnic()` передается словарь `picnicItems`. В нем содержатся переменные, имеющие следующие значения: `sandwiches` — 4, `apples` — 12, `cups` — 4 и `cookies` — 8000. Мы хотим представить информацию в двух столбцах, причем слева должно отображаться название блюда, а справа — количество.

Для этого нужно решить, какой ширины должны быть левый и правый столбцы. Эти значения будут передаваться методу `printPicnic()` вместе со словарем.

Метод `printPicnic()` принимает словарь, а также значения ширины для левого (`leftWidth`) и правого (`rightWidth`) столбцов. Он выводит заголовок `PICNIC ITEMS` над таблицей по ее центру. Затем он обрабатывает в цикле элементы словаря, выводя каждую пару "имя–значение" в отдельной строке таблицы, причем ключ выравнивается влево, значение — вправо, а оставшиеся в каждом столбце пустые позиции заполняются пробелами.

Определив метод `printPicnic()`, мы определяем словарь `picnicItems` и дважды вызываем метод `printPicnic()`, передавая ему различные значения ширины левого и правого столбцов.

В процессе выполнения программа дважды выводит данные. В первый раз ширина левой колонки составляет 12 символов, а правой — 5. Во второй раз эти значения составляют соответственно 20 и 6 символов.

```
---PICNIC ITEMS--
sandwiches.. 4
apples...... 12
cups........ 4
cookies..... 8000
-------PICNIC ITEMS-------
sandwiches.......... 4
apples.............. 12
cups................ 4
cookies............. 8000
```

Используя методы `rjust()`, `ljust()` и `center()`, вы можете быть уверены в том, что данные в строках таблицы аккуратно выровнены, даже если точное количество символов, содержащихся в каждой строке, неизвестно.

Удаление пробелов с помощью методов `strip()`, `rstrip()` и `lstrip()`

Иногда возникает необходимость в удалении из строки ее ведущих, замыкающих или и тех, и других пробельных символов (пробелы, символы табуляции и символы новой строки). Метод `strip()` возвращает новую строку без начальных и конечных пробельных символов. Методы `lstrip()` и `rstrip()` удаляют пробельные символы соответственно в начале и конце строки. Введите в интерактивной оболочке следующие команды.

```
>>> spam = '  Hello World  '
>>> spam.strip()
'Hello World'
>>> spam.lstrip()
'Hello World  '
>>> spam.rstrip()
'  Hello World'
```

С помощью необязательного строкового аргумента можно указать, какие символы должны удаляться с обоих концов строки. Введите в интерактивной оболочке следующие команды.

```
>>> spam = 'SpamSpamBaconSpamEggsSpamSpam'
>>> spam.strip('ampS')
'BaconSpamEggs'
```

Передавая методу `strip()` аргумент `'ampS'`, мы сообщаем ему, что в начале и конце строки, сохраненной в переменной `spam`, должны быть удалены все вхождения символов `a`, `m`, `p` и `S`. Порядок указания символов в строке, передаваемой методу `strip()`, несуществен: вызовы `strip('mapS')` или `strip('Spam')` дадут тот же результат, что и вызов `strip('ampS')`.

Копирование и вставка строк с помощью модуля `pyperclip`

В модуле `pyperclip` имеются функции `copy()` и `paste()`, которые могут выполнять операции копирования и вставки текста через буфер обмена вашего компьютера. Пересылка вывода программы в буфер обмена упрощает вставку текста в сообщения электронной почты или документы текстового процессора, а также его передачу другому программному обеспечению.

Модуль `pyperclip` не поставляется вместе с Python. Чтобы его установить, следуйте указаниям по установке модулей сторонних разработчиков, приведенным в приложении А. Установив модуль, введите в интерактивной оболочке следующие команды.

```
>>> import pyperclip
>>> pyperclip.copy('Hello world!')
>>> pyperclip.paste()
'Hello world!'
```

Разумеется, если содержимое буфера будет изменено внешней программой, то именно оно будет возвращено методом `paste()`. *Например, если я скопирую данное предложение в буфер обмена, а затем вызову метод `paste()`, то получу следующий вывод*:

```
>>> pyperclip.paste()
'Например, если я скопирую это предложение в буфер обмена, а
затем вызову метод paste(), то получу следующий вывод:'
```

Выполнение сценариев Python вне IDLE

До сих пор вы выполняли свои сценарии на Python с помощью интерактивной оболочки или файлового редактора в IDLE. Однако вряд ли вам понравится открывать IDLE всякий раз, когда потребуется выполнить сценарий. К счастью, существуют более удобные способы, упрощающие запуск сценариев Python. Соответствующие процедуры запуска сценариев для Windows, OS X и Linux немного различаются, но все они описаны по отдельности в приложении Б. Загляните в него, если хотите узнать подробнее об этих способах, а также о том, как передавать сценариям аргументы командной строки. (В IDLE такая возможность отсутствует.)

Проект: парольная защита

Возможно, у вас есть учетные записи на многих веб-сайтах. Использовать для каждой из них один и тот же пароль — плохая привычка, поскольку при наличии брешей в системе безопасности любого из этих сайтов хакерам открывается доступ ко всем вашим учетным записям. Наилучшее решение — использовать менеджер паролей на своем компьютере с одним главным паролем для доступа к нему. Открыв менеджер паролей, вы сможете скопировать нужный пароль в буфер обмена и вставить его в поле Введите пароль, предоставляемое при попытке доступа к веб-сайту.

Предлагаемая в этом примере программа менеджера паролей не обеспечивает полной безопасности, но демонстрирует базовые принципы работы программ подобного рода.

Проекты в главах

Это первый из "проектов в главе", которые встречаются в книге. Начиная с этой главы и во всех последующих вам будут предлагаться проекты, демонстрирующие применение на практике понятий, рассмотрению которых была посвящена глава. Особенностью написания всех проектов является то, что каждый из них начинается с "чистого листа" (пустого окна файлового редактора) и заканчивается полностью функциональным вариантом программы. Как и при работе с примерами в интерактивной оболочке, было бы желательно, если бы вы не просто читали эти разделы, а прорабатывали их у себя на компьютере.

Шаг 1. Проектирование программы и структур данных

Мы хотим, чтобы программа принимала аргумент командной строки, которым будет служить имя учетной записи, например учетной записи электронной почты или блога. Пароль для доступа к этой учетной записи будет

копироваться в буфер обмена, откуда пользователь сможет вставить его в поле Password (Пароль). Благодаря этому пользователь не должен держать в памяти длинные пароли, которые надежны, но трудно запоминаются.

Откройте новое окно файлового редактора и сохраните его пустое содержимое в файле *pw.py*. Программа должна начинаться строкой директивы #! (см. приложение Б), за которой следует комментарий, содержащий краткое описание назначения программы. Поскольку вы хотите связывать с именем каждой учетной записи пароль для доступа к ней, целесообразно хранить эти данные в виде строк словаря. Этот словарь и будет той структурой данных, которая обеспечивает организованное хранение имени учетной записи и соответствующего пароля. Введите следующий код.

```
#! python3
# pw.py - Программа для незащищенного хранения паролей.

PASSWORDS = {'email': 'F7minlBDDuvMJuxESSKHFhTxFtjVB6',
             'blog': 'VmALvQyKAxiVH5G8v01if1MLZF3sdt',
             'luggage': '12345'}
```

Шаг 2. Обработка аргументов командной строки

Аргументы командной строки будут храниться в переменной sys.argv. (Более подробная информация о том, как использовать аргументы командной строки в программах, приведена в приложении Б.) Первой в списке sys.argv всегда должна быть строка, содержащая имя файла программы ('pw.py'), а вторым — первый из аргументов командной строки. Таким аргументом для нашей программы является имя учетной записи, с которой вы хотите ассоциировать пароль. Поскольку этот аргумент обязательный, вы отображаете сообщение, напоминающее пользователю о необходимости ввода имени учетной записи, если он забыл его ввести (это будет в том случае, если в списке sys.argv содержится менее двух аргументов). Дополните ранее введенный код следующим.

```
#! python3
# pw.py - Программа для незащищенного хранения паролей.

PASSWORDS = {'email': 'F7minlBDDuvMJuxESSKHFhTxFtjVB6',
             'blog': 'VmALvQyKAxiVH5G8v01if1MLZF3sdt',
             'luggage': '12345'}
import sys
if len(sys.argv) < 2:
    print('Использование: python pw.py [имя учетной записи] -
↳ копирование пароля учетной записи')
    sys.exit()

account = sys.argv[1] # первый аргумент командной строки — это
                      # имя учетной записи
```

Шаг 3. Копирование пароля

Теперь, когда имя учетной записи сохранено в виде строки в переменной account, вы должны проверить, существует ли оно в словаре PASSWORDS в виде ключа. Если существует, вы копируете значение ключа в буфер обмена, используя вызов pyperclip.copy(). (Поскольку используется модуль pyperclip, его необходимо импортировать.) Заметьте, что на самом деле без переменной account можно было бы обойтись, поскольку везде в программе вместо нее можно было бы использовать выражение sys.argv[1]. Однако лучше использовать переменную account, так как читать программу в этом случае гораздо легче.

Дополните ранее введенный код, как показано ниже.

```
#! python3
# pw.py - Программа для незащищенного хранения паролей.

PASSWORDS = {'email': 'F7minlBDDuvMJuxESSKHFhTxFtjVB6',
             'blog': 'VmALvQyKAxiVH5G8v01if1MLZF3sdt',
             'luggage': '12345'}

import sys, pyperclip
if len(sys.argv) < 2:
    print('Использование: python pw.py [имя учетной записи] -
↳ копирование пароля учетной записи')
    sys.exit()

account = sys.argv[1] # первый аргумент командной строки — это
                      # имя учетной записи

if account in PASSWORDS:
    pyperclip.copy(PASSWORDS[account])
    print('Пароль для ' + account + ' скопирован в буфер.')
else:
    print('Учетная запись ' + account + 'отсутствует в списке.')
```

Добавленный код выполняет поиск имени учетной записи в словаре PASSWORDS. Если это имя является ключом в словаре, то мы получаем значение, соответствующее этому ключу, копируем его в буфер обмена и выводим сообщение, которое подтверждает выполнение копирования. В противном случае выводится сообщение о том, что учетная запись с таким именем отсутствует в списке.

Этот сценарий представляет собой полностью завершенную программу. Воспользовавшись приведенными в приложении Б инструкциями по запуску программ из командной строки, вы можете теперь быстро копировать пароли своих учетных записей в буфер обмена. Если вам понадобится обновить программу для изменения или добавления паролей, то достаточно будет внести необходимые изменения в код словаря PASSWORDS.

Разумеется, вряд ли вы захотите хранить все свои пароли в одном месте, откуда любой человек может легко их скопировать. Но можно модифицировать эту программу и использовать ее для быстрого копирования обычного текста в буфер обмена. Предположим, вы отправляете несколько писем электронной почты, в которых содержится много общих абзацев. Вы можете поместить каждый абзац в качестве значения в словарь `PASSWORDS` (в этом случае вы, вероятно, измените имя словаря), а затем будете иметь возможность быстро выбирать и копировать множество стандартных фрагментов текста в буфер обмена.

Пользователи Windows могут создать пакетный файл (файл с расширением *.bat*) для запуска этой программы из окна Выполнить, которое можно легко открыть, нажав комбинацию клавиш <Windows+R>. (Более подробно о пакетных файлах читайте в приложении Б.) Введите в окне файлового редактора и сохраните в файле *pw.bat* в папке *C:\Windows* следующий код.

```
@py.exe C:\Python34\pw.py %*
@pause
```

С использованием только что созданного пакетного файла выполнение программы для хранения паролей сводится к нажатию комбинации клавиш <Windows+R> и вводу команды `pw <имя_учетной_записи>`.

Проект: добавление маркеров в разметку Wiki-документов

Редактируя статьи в Википедии, можно создавать маркированные списки, вводя каждый элемент списка в отдельной строке, которая предваряется маркером в виде звездочки. Предположим, что у вас имеется очень большой список, в который нужно добавить маркеры. Вы могли бы сделать это вручную, вводя символы звездочки в начале каждой строки, и так строка за строкой. Но эту задачу можно легко автоматизировать с помощью короткого сценария Python.

Сценарий *bulletPointAdder.py* будет получать текст из буфера обмена, добавлять звездочку и пробел в начале каждой строки, а затем возвращать этот новый текст в буфер обмена. Например, если бы я скопировал в буфер обмена текст (предназначенный для публикации в Википедии статьи “Список списков списков”)

```
Lists of animals
Lists of aquarium life
Lists of biologists by author abbreviation
Lists of cultivars
```

а затем выполнил программу *bulletPointAdder.py*, то в буфере обмена содержался бы следующий текст:

```
* Lists of animals
* Lists of aquarium life
* Lists of biologists by author abbreviation
* Lists of cultivars
```

Этот предваряемый звездочками текст полностью подготовлен к вставке в статью Википедии в качестве маркированного списка.

Шаг 1. Копирование и вставка посредством буфера обмена

Вы хотите, чтобы программа *bulletPointAdder.py* делала следующее:

1) вставляла текст из буфера обмена;
2) выполняла над ним некоторые действия;
3) копировала новый текст в буфер обмена.

С пунктом 2 придется немного повозиться, а вот пункты 1 и 3 достаточно просты: они требуют использования всего лишь двух вызовов — `pyperclip.copy()` и `pyperclip.paste()`. Поэтому на данном этапе мы напишем только ту часть программы, которая реализует шаги 1 и 3. Введите в файловом редакторе приведенный ниже код и сохраните его в файле *bulletPointAdder.py*.

```
#! python3
# bulletPointAdder.py - Добавляет маркеры Википедии в начало
# каждой строки текста, сохраненного в буфере обмена.

import pyperclip
text = pyperclip.paste()

# TODO: разделить строки и добавить звездочки.

pyperclip.copy(text)
```

Комментарий `TODO` (ЧТО СДЕЛАТЬ) — напоминание о том, что эту часть программы еще предстоит написать. Следующий шаг — фактическая реализация данной части программы.

Шаг 2. Разбивка текста на строки и добавление звездочек

Вызов `pyperclip.paste()` возвращает весь текст из буфера в виде одной большой строки. Если бы мы использовали пример со статьей “Список списков”, то строка, сохраненная в виде текста, выглядела бы так:

```
'Lists of animals\nLists of aquarium life\nLists of biologists by
author abbreviation\nLists of cultivars'
```

Наличие в этой строке символов новой строки \n приведет к тому, что она будет отображаться в виде многострочного текста при выводе на экран или вставке из буфера обмена. В этом одном строковом значении содержится много "строк". Вам нужно добавить звездочку в начале каждой из этих строк.

Можно было бы написать код, который выполняет поиск всех символов \n и вставляет после каждого из них звездочку. Однако гораздо проще использовать метод split() для возврата списка строк, по одной для каждой строки оригинального текста, а затем добавить звездочку в начале каждой строки, входящей в список. Придайте программе следующий вид.

```
#! python3
# bulletPointAdder.py - Добавляет маркеры Википедии в начало
# каждой строки текста, сохраненного в буфере обмена.

import pyperclip
text = pyperclip.paste()

# Разделяет строки и добавляет звездочки.
lines = text.split('\n')
for i in range(len(lines)):     # цикл по списку "lines"
    lines[i] = '* ' + lines[i]  # добавляет звездочку в каждую
                                # строку в списке "lines"

pyperclip.copy(text)
```

Выполняя разбиение текста по символам новой строки, мы получаем список, в котором каждый элемент списка располагается в отдельной строке текста. Мы сохраняем этот список в строках, а затем проходим в цикле по всем элементам в этих строках. В начале каждой строки мы добавляем звездочку и пробел. Теперь каждая текстовая строка начинается со звездочки.

Шаг 3. Объединение измененных строк

Теперь строки списка содержат измененные строки, начинающиеся со звездочек. Но метод pyperclip.copy() ожидает значение в виде одиночной строки, а не списка строковых значений. Чтобы создать это одиночное строковое значение, передайте строки методу join() для их объединения в одну строку. Дополните программный код, как показано ниже.

```
#! python3
# bulletPointAdder.py - Добавляет маркеры Википедии в начало
# каждой строки текста, сохраненного в буфере обмена.

import pyperclip
text = pyperclip.paste()
```

```
# Разделяет строки и добавляет звездочки.
lines = text.split('\n')
for i in range(len(lines)):     # цикл по списку "lines"
    lines[i] = '* ' + lines[i]  # добавляет звездочку в каждую
                                # строку в списке "lines"
text = '\n'.join(lines)
pyperclip.copy(text)
```

В процессе выполнения программа заменяет текст из буфера обмена текстом, в начале каждой строки которого располагаются звездочки. Этот код представляет завершенный вариант программы, и вы можете попытаться выполнить ее в отношении текста, скопированного в буфер обмена.

Даже если вы не испытываете потребности в автоматизации этой довольно специфической задачи, вам могут понадобиться другие виды манипуляций с текстом, такие как удаление замыкающих пробелов в конце строк или преобразование текста в верхний или нижний регистр. Какой бы ни была специфика ваших запросов, вы можете использовать буфер обмена для выполнения операций ввода и вывода.

Резюме

Текст — распространенная форма представления данных, и Python поставляется с множеством полезных методов, предназначенных для обработки текста, сохраненного в строковых значениях. Вы можете использовать индексирование, извлечение срезов и строковые методы практически в любой программе на Python, которую будете писать.

Программы, которые вы сейчас пишете, кажутся не слишком сложными: они не имеют графического интерфейса пользователя с красочными изображениями и цветным текстом. Пока что вы отображаете текст с помощью метода `print()` и предоставляете пользователю возможность вводить текст, используя метод `input()`. Однако пользователь может ускорить ввод больших объемов текста посредством буфера обмена. Эта возможность открывает широкий простор для написания программ, манипулирующих большими объемами текстовых данных. Пусть даже эти программы не содержат окон или графики, но они могут быстро выполнять массу полезной работы.

Другой способ манипулирования большими объемами текста — это чтение и запись текста путем непосредственного обмена данными с жестким диском. О том, как это делается с помощью Python, вы узнаете из следующей главы.

Контрольные вопросы

1. Что такое экранированные символы?
2. Что представляют собой экранированные символы `\n` и `\t`?
3. Как добавить символ обратной косой черты (`\`) в строку?
4. Строковое значение `"Howl's Moving Castle"` — это допустимая строка. Почему она не вызовет ошибку, несмотря на наличие неэкранированного символа апострофа в слове `Howl's`?
5. Если вы не хотите вставлять символ `\n` в свою строку, то как вы напишете строку, содержащую символы новой строки?
6. Каковы будут результаты вычисления приведенных ниже выражений?
   - `'Hello world!'[1]`
   - `'Hello world!'[0:5]`
   - `'Hello world!'[:5]`
   - `'Hello world!'[3:]`
7. Каковы будут результаты вычисления приведенных ниже выражений?
   - `'Hello'.upper()`
   - `'Hello'.upper().isupper()`
   - `'Hello'.upper().lower()`
8. Каковы будут результаты вычисления приведенных ниже выражений?
   - `'Remember, remember, the fifth of November.'.split()`
   - `'-'.join('There can be only one.'.split())`
9. Какие методы используются для выравнивания строки по левому краю, правому краю и центру?
10. Как удалить пробельные символы в начале и конце строки?

Учебный проект

Чтобы закрепить полученные знания на практике, напишите программу для предложенной ниже задачи.

Табличный вывод данных

Напишите функцию `printTable()`, которая принимает список списков строк и отображает его в виде аккуратной таблицы с выравниванием текста по правому краю в каждом столбце. Исходите из предположения, что внутренние списки будут содержать одно и то же количество строк. Например, список мог бы выглядеть так.

```
tableData = [['apples', 'oranges', 'cherries', 'banana'],
             ['Alice', 'Bob', 'Carol', 'David'],
             ['dogs', 'cats', 'moose', 'goose']]
Manipulating Strings 143
```

Вывод функции `printTable()` будет примерно таким.

```
  apples Alice  dogs
 oranges   Bob  cats
cherries Carol moose
  banana David goose
```

Подсказка. Ваш код должен в первую очередь найти самую длинную строку в каждом из внутренних списков; ширина столбца должна быть достаточно большой для того, чтобы в нем уместилась любая строка. Значения максимальной ширины столбцов могут храниться в виде списка целых чисел. Код функции `printTable()` может начинаться с инструкции `colWidths = [0] * len(tableData)`, которая создает список, содержащий значения `0` в количестве, равном количеству внутренних списков в списке `tableData`. Таким образом, в элементе `colWidths[0]` может храниться ширина самой длинной строки, содержащейся в `tableData[0]`, в элементе `colWidths[1]` — ширина самой длинной строки, содержащейся в `tableData[1]` и т.д. Затем вы можете найти самое большое значение в списке `colWidths`, чтобы определить, какое целочисленное значение ширины следует передать строковому методу `rjust()`.