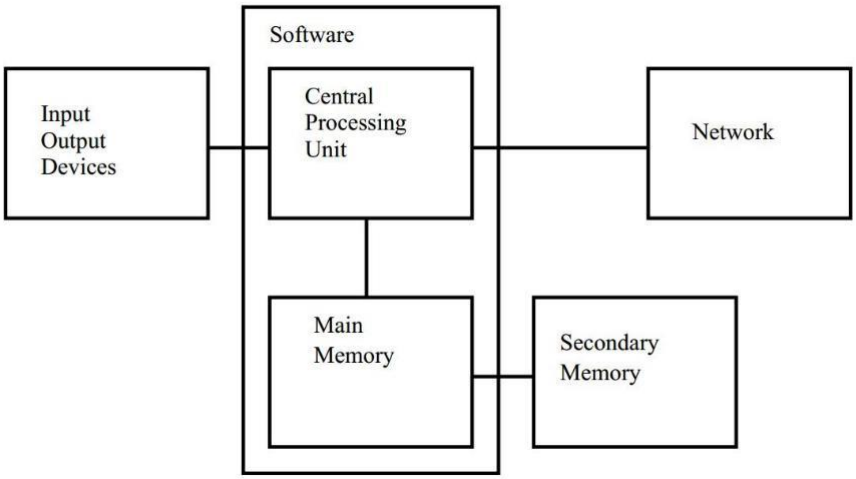
Глава 7. Файлы
7.1. Введение

До сих пор мы учились тому, как писать программы и сообщать о наших намерениях центральному процессору (Central Processing Unit), используя условные инструкции, функции и итерации. Мы учились, как создавать и использовать структуры данных в оперативной памяти (Main Memory). В CPU и памяти работает и выполняется наше программное обеспечение (ПО). Здесь происходит "мышление".

Но если вы помните из нашего обсуждения архитектуры оборудования, одно выключение питания и все хранящееся в CPU или оперативной памяти стирается. До этого момента наши программы имели временное развлекательное назначение для изучения Python.



В этой главе мы начинаем работать с вторичной памятью (Secondary Memory) или файлами. Вторичная память не очищается, когда выключается электричество. В случае с USB-флешками, данные можно на них записать, после этого USB-флешки могут быть удалены из системы и перенесены в другую систему.

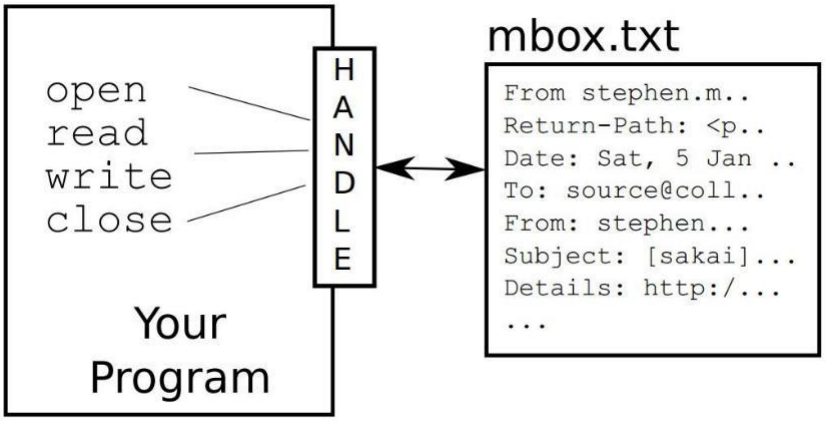
Мы сосредоточимся на чтении и записи текстовых файлов, таких, которые создаются в текстовом редакторе. Позже мы рассмотрим, как работать с файлами баз данных, такими, как бинарные файлы, специально разработанные для чтения и записи через ПО баз данных.

7.2. Открытие файлов

Когда мы хотим прочитать или записать файл (скажем, на жестком диске), во-первых мы должны открыть этот файл. Об открытии файла сообщается операционной системе (ОС), которая знает, где хранятся данные для каждого из файлов. Когда вы открываете файл, вы обращаетесь к ОС, чтобы найти файл по имени и удостовериться, что файл существует. В следующем примере мы открываем файл *mbox.txt*, который должен находиться в той же папке, откуда запускается Python. Скачать файл можно по ссылке: pycode.ru/files/python/mbox.txt

```
>>> fhand = open('mbox.txt')
>>> print fhand
<_io.TextIOWrapper name='mbox.txt' mode='r' encoding='cp1251'>
```

Если функция *open* завершится успешно, то ОС вернет дескриптор файла (file handle). Дескриптор файла не является действующими данными, содержащимися в файле, это "обработчик", который мы можем использовать для чтения данных. У вас есть дескриптор, если запрашиваемый файл существует и имеет надлежащие права на чтение.



Если файл не существует, функция *open* выдаст ошибку с указанием причины и вы не получите дескриптор с доступом к содержимому файла:

```
>>> fhand = open('stuff.txt')
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in
    <module> fhand = open('staff.txt')
IOError: [Errno 2] No such file or directory: 'staff.txt'
```

Позже мы воспользуемся инструкциями *try* и *except*, чтобы сделать более изящной обработку ситуации, когда мы открываем файл, который не существует.

7.3. Текстовый файл и строки

Текстовый файл можно представить как последовательность строк. В качестве примера рассмотрим текстовый файл, который содержит записи почтовой активности от различных лиц в команде разработчиков проекта с открытым исходным кодом:

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org> Date:
Sat, 5 Jan 2008 09:12:18 -0500 To:
source@collab.sakaiproject.org
From: stephen.marquard@uct.ac.za
Subject: [sakai] svn commit: r39772 - content/branches/
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
...
Полная версия файла доступна по ссылке:
pycode.ru/files/python/mbox.txt и сокращенная версия pycode.ru/files/python/mbox-short.txt. Это файлы, которые содержат множество почтовых сообщений в стандартном формате. Строки, которые начинаются с "From " отделяют сообщение и строки, которые начинаются с "From:" являются частью сообщения. Более подробная информация по ссылке: en.wikipedia.org/wiki/Mbox.

Чтобы разбить файл на строки, существует специальный символ, который представляет "конец строки" и называется символом новой строки.
В Python мы представляем символ новой строки, как строковую константу '/n'. Даже не смотря на то, что это выглядит как два символа - на самом деле один символ. Когда мы смотрим переменную, введя "stuff" в интерпретаторе, она отображается с \n в строке, но когда мы используем функцию *print* - видим, что строка разбилась на две по символу новой строки.

```
>>> stuff = 'Hello\nWorld!'
>>> stuff
'Hello\nWorld!'
>>> print
stuff Hello
```

```
  World!
  >>> len(stuff)
  12
>>>
```

Также можно видеть, что длина строки *'Hello\nWorld!'* составляет 12 символов, т.к. новый символ считается за один.

Поэтому, когда мы смотрим на строки в файле, необходимо *представлять* (!), что есть специальные невидимые символы в конце каждой строки, которые обозначают конец строки.

*From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008**n***
*Return-Path: <postmaster@collab.sakaiproject.org>**n** Date:*
*Sat, 5 Jan 2008 09:12:18 -0500**n** To:*
*source@collab.sakaiproject.org**n***
*From: stephen.marquard@uct.ac.za**n***
*Subject: [sakai] svn commit: r39772 - content/branches/**n***
*Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772**n***
...
Таким образом, символ новой строки отделяет символы в файле внутри строк.

7.4. Чтение файлов

До тех пор, пока дескриптор файла не содержит данные, создадим конструкцию с оператором *for*, читая в цикле и увеличивая счетчик для каждой строки в файле.

```
>>> fhand = open('mbox.txt')
>>> count = 0
 >>> for line in fhand:
      count = count + 1


>>> print 'Line Count:',
count Line Count: 132045
```

Мы можем использовать дескриптор файла как последовательность в цикле *for*. Наш цикл считает количество строк в файле и выводит окончательное значение на экран. Грубый перевод цикла *for* на человеческий язык: "для

каждой строки в файле, представленной файловым дескриптором, увеличить переменную *count* на единицу".

Причина, по которой функция *open* не читает весь файл, заключается в том, что файл может иметь гигабайты данных. Функция *open* занимает одинаковое количество времени, независимо от размера файла.
В случае, когда файл читается с использованием цикла *for*, Python заботится о разбиении данных в файле на отдельные строки, используя символ новой строки. Python читает каждую строку, используя новую строку, и включает новую строку, как последний символ в переменную *line* для каждой итерации цикла *for*.

Поэтому цикл *for* читает данные по одной строке за раз, это эффективное чтение и подсчет строк в большом файле без загрузки всей памяти для хранения данных. Программа, написанная выше, может считать строки в файле любого размера, используя немного памяти для чтения каждой строки, подсчета и их сброса.

Если вы знаете что, файл является относительно небольшим по сравнению с размером вашей оперативной памяти, вы можете прочитать весь файл в одну строку, используя метод *read* для дескриптора файла.

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print len(inp)
94626
>>> print inp[:20]
From stephen.marquar
```

В этом примере все содержимое (94626 символов) файла *mbox-short.txt* прочитано непосредственно в переменную *inp*. Мы воспользовались строковым срезом для печати 20 символов строки данных, хранящейся в *inp*.
Помните, что такой способ использования функции *open* возможен только, если файл данных будет удобно помещаться в оперативную память вашего компьютера, иначе используйте циклы *for* или *while*.

7.5. Поиск через файл

Когда вы ищете данные через файл, это очень общий шаблон чтения через файл, игнорируя большинство строк и обрабатывая строки, которые

соответствуют определенному критерию. Мы можем объединить шаблон для чтения файла со строковыми методами, построив простой механизм поиска.

К примеру, если мы хотим прочитать файл и вывести на экран строки, которые начинаются с префикса "From:", мы можем воспользоваться строковым методом *startswith*, выбрав строки с желанным префиксом.

```
>>> fhand = open('mbox-short.txt')
>>> for line in fhand:
     if line.startswith('From:') :
          print line
```

Программа выполняется, получаем следующий результат:

From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
...

Мы получили желаемый результат. Но откуда пустые строки? Это результат невидимости символа новой строки. Каждая строка оканчивается новой строкой, т.к. функция *print* печатает строку из переменной *line*, которая включает новую строку и затем функция *print* добавляет другую новую строку, в результате мы наблюдаем эффект двойного пробела.

Мы могли бы использовать строковый срез, чтобы напечатать все, кроме последнего символа, но простой подход заключается в использовании метода *rstrip*, который удаляет пробелы в правой части строки следующим образом:

```
>>> for line in fhand:
     line = line.rstrip()
     if line.startswith('From:'):
          print line
```

Получим следующий результат:

From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu

From: rjlowe@iupui.edu
From: zqian@umich.edu
...

Вы можете усложнить структуру поиска, воспользовавшись инструкцией *continue*. Основная идея поиска в цикле - вы ищете "интересные" строки и пропускаете "неинтересные". И когда находите интересную строку - выполняете с ней какие-то действия.

Мы можем построить цикл по следующему шаблону, пропуская неинтересные строки:

```
>>> for line in fhand:
     line = line.rstrip()
     if not line.startswith('From:'):
          continue
     print line
```

Вывод программы совпадает с предыдущим.

Мы можем использовать строковый метод *find* для имитации поиска в текстовом редакторе. Так как *find* ищет вхождения строки в другой строке и либо возвращает позицию строки или -1, если строка не найдена, мы можем написать следующий цикл, чтобы показать строки, которые содержат "@uct.ac.za" (то есть письма приходят из университета Кейптауна в Южной Африке):

```
>>> fhand = open('mbox-short.txt')
>>> for line in fhand:
     line = line.rstrip()
     if line.find('@uct.ac.za') == -1 :
          continue
     print line
```

Получили следующий результат:

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
X-Authentication-Warning: nakamura.uits.iupui.edu: apache set sender to stephen.marquard@uct.ac.za using -f
From: stephen.marquard@uct.ac.za
Author: stephen.marquard@uct.ac.za

From david.horwitz@uct.ac.za Fri Jan 4 07:02:32 2008
X-Authentication-Warning: nakamura.uits.iupui.edu: apache set sender to david.horwitz@uct.ac.za using -f
From: david.horwitz@uct.ac.za
Author: david.horwitz@uct.ac.za
r39753 | david.horwitz@uct.ac.za | 2008-01-04 13:05:51 +0200 (Fri, 04 Jan 2008) | 1 line
From david.horwitz@uct.ac.za Fri Jan 4 06:08:27 2008
X-Authentication-Warning: nakamura.uits.iupui.edu: apache set sender to david.horwitz@uct.ac.za using -f
From: david.horwitz@uct.ac.za
...

7.6. Позволим пользователю выбрать имя файла

Если мы захотим изменить имя файла, то придется изменять и код программы. Чтобы избежать постоянного изменения кода, можно просить пользователя вводить имя файла с клавиатуры. Например, можно воспользоваться *raw_input*:

```
fname = raw_input('Enter the file name: ')
fhand = open(fname)
count = 0
for line in fhand:
     if line.startswith('Subject:') :
          count = count + 1
print 'There were', count, 'subject lines in', fname
```

7.8. Запись файлов

Чтобы записать что-то в файл, необходимо открыть его в режиме ‘*w*’:

```
>>> fout = open('output.txt', 'w')
>>> print fout
<open file 'output.txt', mode 'w' at 0xb7eb2410>
```

Если файл уже существует, то открытие его в режиме записи очистит его и начнет запись заново, будьте внимательны! Если файл не существует, то он будет создан.

Метод *write* файлового объекта помещает данные в файл.

```
>>> line1 = 'This here's the wattle,\n'
>>> fout.write(line1)
```

Кроме того, файловый объект сохраняет путь записи. Если вы снова вызовете метод *write*, то новые данные будут добавлены в конец.

Инструкция *print* автоматически добавляет символ новой строки, метод *write* не добавляет новую строку автоматически.

```
>>> line2 = 'the emblem of our land.\n'
>>> fout.write(line2)
```

Когда вы завершили запись файлов, то необходимо закрыть файл.

```
>>> fout.close()
```

7.10 Словарь

Перехват (catch): To prevent an exception from terminating a program using the try and except statements.

Новая строка (newline): A special character used in files and strings to indicate the end of a line.

Текстовый файл (text file): A sequence of characters stored in permanent storage like a hard drive.

7.11. Упражнение

1. Перечислите основные этапы при работе с файлами в Python.