§ 68. Работа с файлами

Как работать с файлами?

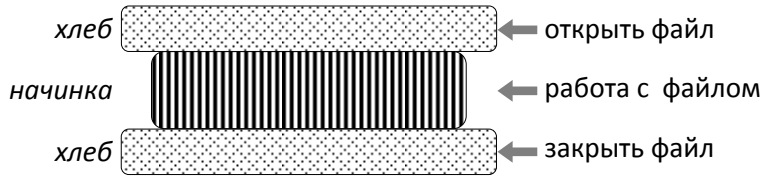
Файл – это набор данных на диске, имеющий имя. С точки зрения программиста, бывают файлы двух типов:

1) *текстовые*, которые содержат текст, разбитый на строки; таким образом, из всех специальных символов в текстовых файлах могут быть только символы перехода на новую строку;
2) *двоичные*, в которых могут содержаться любые данные и любые коды без ограничений; в двоичных файлах хранятся рисунки, звуки, видеофильмы и т.д.

Мы будем работать только с текстовыми файлами.

Работа с файлом из программы включает три этапа. Сначала надо *открыть файл*, то есть сделать его активным для программы. Если файл не открыт, то программа не может к нему обращаться. При открытии файла указывают режим работы: чтение, запись или добавление данных в конец файла. Чаще всего открытый файл блокируется так, что другие программу не могут использовать его. Когда файл открыт (активен) программа выполняет все необходимые операции с ним. После этого нужно *закрыть файл*, то есть освободить его, разорвать связь с программой. Именно при закрытии все последние изменения, сделанные программой в файле, записываются на диск.

Такой принцип работы иногда называют «принципом сэндвича», в котором три слоя: хлеб, затем начинка, и потом снова хлеб:



В большинстве языков программирования с файлами работают через вспомогательные переменные (их называют указатели, идентификаторы и т.п.). В Python есть функция **`open`**, которая открывает файл и возвращает файловый указатель – переменную, через которую идёт вся даль-

нейшая работа с файлом. Функция **open** принимает два параметра: имя файла (или путь к файлу, если файл находится не в том каталоге, где записана программа) и режим открытия файла:

- **"r"** – открыть на чтение,
- **"w"** – открыть на запись,
- **"a"** – открыть на добавление.

Метод **close**, определённый для файлового указателя, закрывает файл:

```
Fin = open ( "input.txt" )
Fout = open ( "output.txt", "w" )
   # здесь работаем с файлами
Fout.close()
Fin.close()
```

При первом вызове функции **open** режим работы с файлом не указан, но по умолчанию предполагается режим **"r"** (чтение).

Если файл, который открывается на чтение, не найден, возникает ошибка. Если существующий файл открывается на запись, его содержимое уничтожается.

После окончания работы программы все открытые файлы закрываются автоматически.

Чтение одной строки из текстового файла выполняет метод **readline**, связанный с файловой переменной:

```
s = Fin.readline()
```

Если нужно прочитать несколько данных в одной строке, разделённых пробелами, используют метод **split**. Этот метод разбивает строку по пробелам и строит список из соответствующих «слов»:

```
s = Fin.readline().split()
```

Если в прочитанной строке файла были записаны числа 1 и 2, список **s** будет выглядеть так:

```
["1", "2"]
```

Элементы этого списка – символьные строки. Поэтому для того, чтобы выполнять с этими данными вычисления, их нужно перевести в числовой вид с помощью функции **int**, применив её для каждого элемента списка:

```
a, b = int(s[0]), int(s[1])
```

То же самое можно записать с помощью генератора

```
a, b = [int(x) for x in s]
```

или с помощью функции **map**, которая применяет функцию **int** ко всем элементам списка:

```
a, b = map( int, s )
```

Запись **map(int,s)** означает «применить функцию **int** ко всем элементам списка **s**».

Для вывода строки в файл применяют метод **write**. Если нужно вывести в файл числовые данные, их сначала преобразуют в строку, например, так:

```
Fout.write ( "{:d} + {:d} = {:d}\n".format(a, b, a+b) )
```

Таким образом мы записали в файл результат сложения двух чисел. Обратите внимание, что, в отличие от функции **print**, при таком способе записи символ перехода на новую строку «**\n**» автоматически не добавляется, и мы добавили его в конце строки вручную.

Как правило, текстовый файл – это «устройство» последовательного доступа к данным. Это значит, что для того, чтобы прочитать 100-е по счёту значение из файла, нужно сначала прочитать предыдущие 99. В своей внутренней памяти система хранит положение указателя (файлового курсора), который определяет текущее место в файле. При открытии файла указатель устанавливается в самое начало файла, при чтении смещается на позицию, следующую за прочитанными данными, а при записи – на следующую свободную позицию.

Если нужно повторить чтение с начала файла, можно закрыть его, а потом снова открыть.

Неизвестное количество данных

Предположим, что в текстовом файле записано в столбик неизвестное количество чисел и требуется найти их сумму. В этой задаче не нужно одновременно хранить все числа в памяти (и не нужно выделять массив!), достаточно читать по одному числу и сразу его обрабатывать:

```
while не конец файла:
  прочитать число из файла
```

```
    добавить его к сумме
```

Для того, чтобы определить, когда данные закончились, будем использовать особенность метода **readline**: когда файловый курсор указывает на конец файла, метод **readline** возвращает пустую строку, которая воспринимается как ложное логическое значение:

```
while True:
  s = Fin.readline()
  if not s: break
```

В этом примере при получении пустой строки цикл чтения заканчивается с помощью оператора **break**.

Возможны и другие варианты. Например, метод **readlines** позволяет прочитать все строки сразу в список:

```
Fin = open ( "input.txt" )
lst = Fin.readlines()
for s in lst:
  print ( s, end = "" )
Fin.close()
```

Строки файла читаются в список **lst** и выводятся на экран. Обратите внимание, что переход на новую строку в функции **print** отключен (**end=""**), потому что при чтении символы перевода строки «**\n**» в конце строк файла сохраняются.

В Python есть ещё один способ работы с файлами, при котором закрывать файл не нужно, он закроётся автоматически. Это конструкция **with-as**:

```
with open ( "input.txt" ) as Fin:
  for s in Fin:
    print ( s, end = "" )
```

В первой строке файл **input.txt** открывается в режиме чтения и связывается с файловым указателем **Fin**. Затем в цикле перебираются все строки в этом файле, каждая из них по очереди попадает в переменную **s** и выводится на экран. Закрывать файл с помощью close не нужно, он закроется автоматически после окончания цикла.

Наконец, приведём ещё один способ в стиле Python:

```
for s in open ( "input.txt" ):
  print ( s, end = "" )
```

Этот вариант обычно рекомендуют к использованию, при нём также не нужно закрывать файл.

Вернёмся к исходной задаче сложения чисел, записанных в файле в столбик. Далее будем считать, что файловая переменная **Fin** связана с файлом, открытым на чтение. Основная часть программы (без команд открытия и закрытия файлов) может выглядеть, например, так:

```
sum = 0
while True:
  s = Fin.readline()
  if not s: break
  sum += int(s)
```

или так:

```
sum = 0
for s in open ( "input.txt" ):
  sum += int(s)
```

Обработка массивов

В текстовом файле записаны целые числа. Требуется вывести в другой текстовый файл те же числа, отсортированные в порядке возрастания.

Особенность этой задачи в том, что для сортировки нам нужно удерживать в памяти все числа, то есть для их хранения нужно выделить массив (список).

В большинстве языков программирования память под массив нужно выделить заранее, поэтому необходимо знать наибольшее возможное число элементов массива. Поскольку список в Python может расширяться, у нас этой проблемы нет. Цикл чтения может выглядеть так:

```
A = []
while True:
```

```
  s = Fin.readline()
  if not s: break
  A.append ( int(s) )
```

Есть и другой вариант, в стиле Python. Метод **read** для файлового указателя читает (в отличие от **readline**) не одну сроку, а весь файл. Затем мы разобьём получившуюся символьную строку на слова с помощью метода **split**:

```
s = Fin.read().split()
```

и преобразуем слова в числа, составив из них список:

```
A = list ( map(int, s) )
```

Теперь нужно отсортировать массив **A** (этот код вы уже можете написать самостоятельно) и вывести их во второй файл, открытый на запись:

```
Fout = open ( "output.txt", "w" )
Fout.write ( str(A) )
Fout.close()
```

В этом варианте мы использовали функцию **str**, которая возвращает символьную запись объекта, такую, которая выводится на экран. Если нам нужно форматировать данные по своему, например, вывести их в столбик, можно обработать каждый элемент вручную в цикле:

```
for x in A:
  Fout.write ( str(x)+"\n" )
```

К символьной записи очередного элемента массива мы добавляем символ перехода на новую строку, который обозначается как **"\n"**. В этом случае числа выводятся в файл в столбик.

Обработка строк

Как известно, современные компьютеры бóльшую часть времени заняты обработкой символьной, а не числовой информации. Предположим, что в текстовом файле записаны данные о собаках, привезенных на выставку: в каждой строчке кличка собаки, ее возраст (целое число) и порода, например,

Мухтар 4 немецкая овчарка

Все элементы отделяются друг от друга одним пробелом. Нужно вывести в другой файл сведения о собаках, которым меньше 5 лет.

В этой задаче данные можно обрабатывать по одной строке (не нужно загружать все строки в оперативную память):

```
while не конец файла (Fin):
  прочитать строку из файла Fin
  разобрать строку - выделить возраст
  if возраст < 5:
    записать строку в файл Fout
```

Здесь, как и раньше, **Fin** и **Fo**ut – файловые переменные, связанные с файлами, открытыми на чтение и запись соответственно.

Будем считать, что все данные корректны, то есть первый пробел отделяет кличку от возраста, а второй – возраст от породы. Тогда для разбора очередной прочитанной строки **s** можно использовать метод **split**, который вернет список, где второй по счёту элемент (он имеет индекс 1) – это возраст собаки. Его и нужно преобразовать в целое число:

```
s = Fin.readline()
data = s.split()
sAge = data[1]
age = int ( sAge )
```

Эти операции можно записать в краткой форме:

```
s = Fin.readline()
age = int ( s.split()[1] )
```

Полная программа (без учёта команд открытия и закрытия файлов) выглядит так:

```
while True:
  s = Fin.readline()
  if not s: break
  age = int ( s.split()[1] )
```

```
    if age < 5:
      Fout.write ( s )
```

Её можно записать несколько иначе, используя метод **readlines**, который читает сразу все строки в список:

```
lst = Fin.readlines()
for s in lst:
  age = int ( s.split()[1] )
  if age < 5:
    Fout.write ( s )
```

или конструкцию **with-as**:

```
with open ( "input.txt" ) as Fin:
  for s in Fin:
    age = int ( s.split()[1] )
    if age < 5:
      Fout.write ( s )
```

Вот ещё один вариант в стиле Python, возможно, наилучший:

```
for s in open ( "input.txt" ):
  age = int ( s.split()[1] )
  if age < 5:
    Fout.write ( s )
```