*Тема 3.1. Работа с файлами.* ***Лекционное занятие (1 час)***

Файл – именованная область на носителе. Организация их хранения зависит от операционной системы. Сейчас наиболее популярна древовидная организация хранения файлов с *логической* привязкой к так называемым «каталогам». Реальная организация

|  |  |
| --- | --- |
| хранения связана с физическим, а не логическим адресом файла на носителе, но это выходит за рамки данного курса.**Рис.1.** Пример древа файлов Windows |  |

Каждый файл обладает различным набором свойств в зависимости от файловой системы. Большинство файловых систем предполагают наличие следующих свойств:

* имя и расширение (вместе – «название»): например, file1.py;
* дата/время (создания, модификации, последнего доступа);
* атрибуты (форма организации и доступа: скрытый, каталог, системный и др.).

Отметим, что на название файла могут быть наложены ограничения в зависимости от файловой и операционной системы, в частности, допустимые знаки и размер компонентов, запрет определенных имен. Расширение, указываемое после имени через точку, используют, например, в Windows, при выборе приложения для его загрузки.

Для доступа к файлу нужен **путь** – список узлов дерева (папок), задающих логический адрес. Вид записи пути также зависит от ОС. Пример: **C:\python\example1.py**

При работе с файлами важно учитывать их виды и способы организации доступа: последовательный или прямой (произвольный).

Файлы с произвольным доступом хранят информацию в структурированном для быстрого доступа к его компонентам виде. Поиск в таких файлах осуществляется в области адресов (ключей) за **O(1)** Условный вид такого файла представлен на рис.2.

|  |  |
| --- | --- |
| Рис.2. Вид файла прямого доступа |  |

Файлы последовательного доступа хранят информацию в неструктурированном для быстрого доступа виде. Поиск здесь идет последовательным считыванием файла с начала и сравнением с искомым за **O(n)**. Обращение к определённому участку требует «чтения с начала», или же считывания данных в какие-либо структуры с оптимизированным доступом к нужным данным, что требует **O(n)** памяти.

|  |  |
| --- | --- |
| Рис.3. Вид файла последовательного доступа |  |

В Python работа с файлами осуществляется через файловый объект. Стандартный способ его создания – вызов функции open(), открывающей его в режиме *файлового потока*, который обеспечивает буферизированную или побайтную обработку данных в зависимости от параметров команды open(). Размер буфера чтения зависит от категории и способа открытия файла. Если буфер не указан, то он имеет размер по умолчанию, узнать который можно следующей программой:

**import** io; **print**(io.DEFAULT\_BUFFER\_SIZE)

*При чтении как файл можно использовать любой поток данных*.

Есть 3 категории файловых объектов (подробнее об этом – см. описание модуля **io**):

* **необработанные** (raw) двоичные файлы/потоки – нет буферизации, размер не всегда кратен байту, т.к. поток может быть из неформатированного/сбойного источника;
* **буферизованные двоичные файлы** (bytes) – размер кратен байту, обычно используют для структур записей, байтовых потоков, \*.exe – файлов и т.п.;
* **текстовые файлы**(lines)–набор строк заданной кодировки и символами завершения.

Каждому файловому объекту назначен прямой указатель места в файле или дескриптор устройства записи/чтения, то есть указатель может быть привязан к любому устройству.

У функции **open**() много параметров, включая, например, размер буфера, но обычно используются: file, mode, encoding, а остальные имеют значение по умолчанию:

* file – путь к файлу или целочисленный дескриптор источника/приемника данных;
* encoding – строка, соответствующая кодировке **текстового** файла;
* mode – режимы открытия файла (некоторые режимы можно комбинировать – 'rb'):

|  |  |  |  |
| --- | --- | --- | --- |
| **Режим** | **Открыть…** | **Режим** | **Открыть…** |
| **'r'** | для чтения (по умолчанию) | **'w'** | для записи |
| **'a'** | для добавления или создания нового | **'b'** | Двоичный режим |
| **'t'** | текстовый режим (по умолчанию) | **'+'** | для чтения и записи |
| **'x'** | Создать и открыть с эксклюзивными правами (ошибка, если уже есть) |

Примеры: a)f = **open**('text.txt'); b)f = **open**('text.txt','r') (формы a и b тождественны).

**Чтение из файла**

Функция read() используется для чтения файла, открытого как 'r': file.**read**(size), где size – количество читаемых символов (если не указать, то читается весь файл).

Примеры: a) f = **open**('text.txt','r'); t=f.**read**(); **print**(t).

b) f = **open**('text.txt','r'); t1=f.**read**(1); t2=f.**read**(1); **print**(t1, t2). Результат – см. рис. 3-5:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Рис.3.** Вид файла: |  | **Рис.4.** Результат работы программы a): |  | **Рис.5.** Результат работы программы b): |  |

Функция readline() используется для построчного чтения большого файла (пример a).

Ещё один способ – прочитать файл построчно в цикле for (пример b).

|  |  |  |
| --- | --- | --- |
| a) f = **open**('text.txt'); t1=f.**readline()**; **print**(t1) | **Рис. 6.** Результат a): | **123** |
| b) f = **open**('text.txt'); **for** t **in** f: **print**(t, end=' ') | **Рис. 7.** Результат b): | **123** **456** **789** |

**Запись в файл**

Функця write() пишет в файл, открытый как 'w': file.**write**(строковая переменная).

При открытии файла для записи, существующий будет перезаписан, новый – создан.

Важно, что в качестве аргумента функции могут быть переданы только строки!!!

Пример: f = **open**('f1.txt','w'); f.**write**('Привет')

Можно записать в файл список строк: file.**writelines**(список строк).

Следует обратить внимание, что write() и writelines() не переводят строку (!), а потому при записи следует ставить в конце переход к новой строке: '\n'.

После завершения работы, файл обязательно нужно закрыть: file.**close**().

Примеры: a) f = **open**('f1.txt','w'); f.**write**('Привет'); x=128; f.write(**str**(x)); f.**close**()

b) f = **open**('f1.txt','w'); f.**write**('Привет'+'\n'); x=128; f.**write**(**str**(x)); f.**close**()

Результаты работы программы представлен ниже:

|  |  |  |  |
| --- | --- | --- | --- |
| **Рис.8.** Результат работы программы a: |  | **Рис.9.** Результат работы программы b: |  |

**Запись на консоль как в файл:** **import** sys; f=sys.**stdout**; f.**write**('123'+'\n')

***3.1.Практическое занятие (1 час)***

1. Написать программу для создания и записи в файл numb.txt 5 введенных чисел.

f = **open**('numb.txt','w')

**for** i **in** **range**(5): f.**write**(**input**('Введи '+**str**(i+1)+ '-е число (из 5):')+'\n')

f.**close**() #Обратите внимание, что без f.**close**() изменения не будут сохранены.

1. Написать программу, которая дописывает в файл numb.txt еще 5 целых чисел.

f = **open**('numb.txt','a') # Обратите внимание на атрибут открытия файла

**for** i **in** **range**(5): f.**write**(**input**('Введи '+**str**(i+1)+ '-е число (из 5):')+'\n')

f.**close**()

1. Написать программу вывода и подсчета среднего арифметического файла numb.txt.

f = **open**('numb.txt', 'r'); s=0; c=0; # Внимание! Атрибут открытия файла не обязателен

**for** x **in** f: **print**(x); s+=**int**(x); c+=1

**print**(s/c) #Отметим, что close() при чтении писать тоже не обязательно, но желательно.

1. Написать программу поиска среднего арифметического, минимума и максимума чисел в файле и дописать их в конец файла.

f=**open**('numb.txt','r+');c=1;s=mn=mx=**int**(f.**readline**())#читаем первую строку

**for** x **in** f: #дальше файл будет читаться со второй строки

 m=**int**(x); s+=m; c+=1

 **if** mn>m: mn=m

 **elif** mx<m: mx=m

sr=s/c; f.**write**(**str**(mx)+'\n'+**str**(mn)+'\n'+**str**(sr)); f.**close**()

Заметим, что закрытие чтения и открытиt того же файла для дозаписи – режим 'r+'. После внесения изменений файл следует закрыть.

Если объём данных не очень большой, то для задач 3-4 можно воспользоваться чтением и преобразованием в список целых чисел сразу всего файла:

1. Написать программу вывода и подсчета среднего арифметического файла numb.txt.

f=**open**('numb.txt');x=**list**(**map**(**int**,f.**readlines()**)) #отметим обработку f.readlines()!

**print**(\*x); **print**(**sum**(x)/**len**(x))#здесь close() не нужен, в отличие от следующей задачи

Здесь *используем* **O(n)** *дополнительной* *памяти и повторный проход – подсчет суммы*.

1. Написать программу поиска среднего арифметического, минимума и максимума чисел в файле и дописывания результатов в конец файла.

f = **open**('numb.txt','r+'); x=**list**(**map**(**int**,f.**readlines()**))

**print**(\*x); sr=**sum**(x)/**len**(x); mn=**min**(x); mx=**max**(x)

**print**('max=',mx,'\nmin=',mn,'\nsred=', sr, sep='')

f.**write**(**str**(mx)+'\n'+**str**(mn)+'\n'+**str**(sr)); f.**close()**

 Здесь тоже программа короче, но также *используем* **O(n)** *дополнительной памяти, но уже три повтроных прохода данных для суммы, минимума и максимума!* В то же время, в большинстве задач такое снижение эффективности алгоритма не критично.

 В заключение отметим, что *в работе с файлами можно использовать операторы input() и print()*, переназначив консоль (stdin-клавиатура, stdout-монитор) в файл:

**from** sys **import** stdin, stdout; stdin=**open**('t1.txt'); stdout=**open**('t2.txt', 'w')

x=stdin.**readlines()**; **print**(\*x); stdout.**close()**#ввод из файла, вывод на экран, но в файл.

*Тема 3.2. Работа с файлами.* ***Лекционное занятие (1 час)***

Рассмотрим подробнее режимы доступа в open() при создании нового файла:

* x: создаем новый файл с указанным именем. Если такой уже есть – ошибка;
* a: создаем новый файл, если его нет, или дописываем к существующему;
* w: создаем новый файл, если такого нет или перезаписывам существующий.

Любой, даже текстовый, файл одновременно и двоичный[[1]](#footnote-1), но в текстовом файле символы закодированы посредством какой-либо схемы/стандарта (ASCII, Юникод…).

В общем случае данные бинарного файла нельзя преобразовать в текст. Здесь используют строки типа bytes. Такие файлы просматривают в соответствующих программах, и даже в некоторых текстовых редакторах, если там имеется режим бинарного просмотра/редактирования. Примеры открытия файла в бинарном режиме:

f1 = **open**('file1.bin', 'wb');f2 = **open**('file2.bin', 'w+b').

В отличие от текстовых, бинарные файлы не выполняют преобразования символов конца строки '\n'. Рассмотрим пример создания/записи бинарного файла:

L = [1, 2, 3,8,-6,5,0]; f = **open**('file3.bin', 'wb')

**for** x **in** L: s=; bs=s.**encode()**; f.**write**(bs)#можно также: f.**write**((**str**(x)+'\n').**encode()**)

f.**close()**

Метод encode() конвертирует строку в цепочку байт. Рассмотрим процесс чтения:

|  |  |
| --- | --- |
| f = **open**('file3.bin', 'rb'); d = f.**read**(); **print**(d)**Рис.10.** Результат работы программы: |  |

Если выводить посимвольно, то получим целочисленный код символа:

|  |  |
| --- | --- |
| f = **open**('file3.bin', 'rb'); d = f.**read**(); **print**(d)**print**("d[5] = ", d[5]); **print**("d[0] = ", d[0])**Рис.11.** Результат работы программы: |  |

Анализ рассмотренных примеров позволяет сделать следующие выводы:

* при выводе строка бинарных данных выводится как строка;
* при посимвольном выводе получаем 8-битные целые числа – ASCII-коды символов.

В работе с бинарными файлами слеует рассмотреть ***сериализацию*** – процесс перевода какой-либо структуры данных в последовательность битов. ***Десериализация*** представляет собой обратный процесс. Чаще всего сериализация используется для сохранения объектов в файлы или передачи их в потоке, например, по сети.

Одним из способов, позволяющих сериализовать/десериализовать данные в Python, является использование стандартного модуля pickle, при помощи которого можно сохранять любой объект Python в двоичном файле, а затем извлекать его обратно.

Рассмотрим основные функции данного модуля на примере сериализации словаря.

**dump**(obj,fL,protocol=None, \*,fix\_imports=True)–cериализует obj и пишет в файл fL.

Параметры: 1)obj–объект для записи; 2)fL–файловый объект; 3)protocol–версия pickle.

**load**(fL, \*, fix\_imports=True, encoding='ASCII', errors='strict') – читает и десериализует данные файлового объекта fL, возвращая созданный объект (структуру). Параметры:

1)fL – объект;2)encoding – кодировка;3)errors – метод контроля ошибок декодирования.

**import** pickle

sh = {"фрукты": ["яблоки", "манго"],"овощи": ["морковь"],"бюджет": 1000}

fh=**open**('f1.bin', "wb"); pickle.**dump**(sh, fh); sh = [] # помещаем объект в файл

**with** **open**('f1.bin', 'rb') **as** fh: sh = pickle.**load**(fh)

**print**(sh\_list\_2)

Созданный бинарный файл f1.bin – см. на рис.12 (везде просмотр через Far-manager).

|  |  |
| --- | --- |
| **Рис.12.** Вид бинарного файла: |  |
| **Рис.13.** Результат работы программы: |  |

К минусам формата [pickle](https://www.yuripetrov.ru/edu/python/ch_08_01.html#module-pickle) можно отнести следующее:

* только для Python (нельзя расшифровать в других языках программирования);
* незащищен (десериализация готовых конструкций языка снижает надежность кода).

Заслуживает внимания побайтная сериализация, где часто используют библиотеку BytesIO, хотя возможно обойтись и без нее. На рис.14 представлен результат работы программы, генерирующей случайный набор из 30 байтов, записываемых в файл:

|  |  |
| --- | --- |
| **from** io **import** BytesIO **as** BIO**from** random **import** randint **as** rndf = **open**('numb1.txt', 'wb')s=[**chr**(**rnd**(0,255)) **for** i **in** **range**(30)]**for** i **in** s: f.**write**(**BIO**(i.**encode()**).**getbuffer**())f.**close**() | **Рис.14**. Результат работы программы |
| **from random import randint as rnd #**Аналогичный результат, но без BytesIO**f =open('numb1.txt', 'wb'); s=[rnd(0,255) for i in range(30)];f.write(bytes(s));f.close()** |

***3.2.Практическое занятие (1 час)***

1. Для введенной строки чисел написать программу их вывода в файл f1.txt построчно.

s=**input**().**split**(' '); f=**open**('f1.txt', 'w')

**for** x **in** s: f.**write**(x+"\n")

f.**close**()

1. Написать программу чтения в список чисел файла f1.txt, вычисления их суммы и максимума, а затем дописывания их в файл f1.txt.

f=**open**('f1.txt', 'r+'); s=**list**(**map**(**int**,f.**readlines()))**

f.**write**(**str**(**sum**(s))+'\n'+**str**(**max**(s))+'\n'); f.**close**()

Обратите внимание, что из файла читаем строки, преобразуя в список целых чисел.

1. Написать программу вывода на экран текстового файла и подсчета максимума количеств слов, начинающихся на гласную и согласную букву (регистр не учитывать).

f=**open**('11.txt'); s=f.**read**();cg=cs=0;**print**(s); s=s.**lower**().**split**()

**for** w **in** s:#каждую строку

 **if** w[0] **in** 'ёуеыаоэяию': cg+=1 #если первая буква гласная

 **elif** w[0] **in** 'йцкнгшщзхъфвпрлджчсмтьб': cs+=1#если первая согласная

**if** cg>cs: **print**('больше, где первая гласная')

**elif** cg<cs: **print**('больше, где первая согласная')

**else**: **print**('одинаково')

1. Написать программу, считывающую из файла данные о занятости мест в зрительном зале кинотеатра (0 – свободно, 1 – занято) и определите, сколько мест свободно.

|  |  |
| --- | --- |
| f=**open**('11.txt'); **print**(f.**read**().**count**('0')) | Рисунок к задаче: |

Здесь все строки читаются в одну строку, где считаем '0'

*Тема 3.3. Работа с файлами.* ***Лекционное занятие: разбор типовых задач (1 час)***

Рассмотрим несколько задач, возникающих при работе с текстовыми файлами.

Пример 1. Тестирование конца файла.

В языке Python для чтения строки используется метод readline(). Если он возвращает пустую строку, то достигнут конец файла (считаюься и пустые строки).

|  |  |  |
| --- | --- | --- |
| f = **open**('11.txt'); c = 0; s = f.**readline**() **while** s != '': s = f.**readline**(); c+=1f.**close**(); **print**('count = ', c) | **Рис. 15**. Исходный файл | **Рис.16.** Результат  |

Пример 2. Подсчет количества строк текстового файла.

Для чтения всех строк файла можно предложить два основных способа:

A)чтение методом readlines() в список. Длина списка=количеству строк;

B)чтение в цикле методом readline() до пустой строки со счетчиком.

A)f = **open**('11.txt'); **print**('count = ', **len**(f.**readlines**()))

B) f = **open**('11.txt'); c = 0;

 **for** s **in** f: c+=1

 **print**('count = ', c)

Пример 3. Замена строки в текстовом файле.

Здесь необходимо:1)Считать файл в список; 2)Изменить строку; 3)Записать всё в файл.

f = **open**('11.txt'); L = f.**readlines**(); s='1 2 3'; p=1

**if** (p >= 0) **and** (p < **len**(L)): L[p] = s + '\n' #строка перезаписывается

f.**close**(); f = **open**('11.txt', 'w')

**for** x **in** L: f.**write**(x)

f.**close**()

Пример 4. Удаление строки из файла по ее индексу.

f = **open**('11.txt'); L = f.**readlines**() #p-номер удаляемой строки

**if** (p >= 0) **and** (p < **len**(L)): L=L[:p]+L[p+1:]

f.**close**(); f = **open**('11.txt', 'w');

**for** x **in** L: f.**write**(x)

f.**close**()

Пример 5. Вставка строки в файл (если адрес строки>последней строки, пишем в конец).

f = **open**('11.txt'); L = f.**readlines**(); s='2 2 2'; p=2

**if ord**(L[-1][-1])!=10: L[-1]+= '\n' #после последней строки нужен '\n'

**if** p<0: f.**close**(); **exit**()

**if** p>=**len**(L): L[-1]+= [s] + '\n'

**else**: L = L[:p]+[s + '\n']+L[p:]

f.**close**(); f = **open**('11.txt', 'w')

**for** x **in** L: f.**write**(x)

f.**close**()

Если у файла в конце пустые строки, то при чтении, например, целых чисел контроль конца данных по примеру 1 не сработает и нужно пользоваться **try**…**except**:

Пример 6. Вычисление суммы чисел, записанных построчно в текстовом файле.

f = **open**('11.txt'); s=0

**try**: #включаем обработку исключений при чтении файла и преобразовании к float

 **while** **True**: s+=**float**(f.**readline**()) #читаем как float в бесконечном цикле, ищем сумму

**except**: f.**close**(); **print**(s) #except – ошибка чтения (конец данных) или преобразования

***3.3.Практическое занятие (1 час)***

1. Написать программу вставки в заданную строку файла введенной строки.

f = **open**('11.txt'); L = f.**readlines**(); s=**input**('Срока: '); p=**int**(**input**('Позиция: '))

**if** (p<0): f.**close**(); **exit**() #ошибка позиции вставки

**if** p>=**len**(L): L[-1] += '\n' + [s] #если позиция больше, чем строк – пишем в конец

**else**: L = L[:p] + [s + '\n'] + L[p:]#можно использовать insert

f.**close**(); f = **open**('11.txt', 'w');f.**writelines**(L);f.**close**()

1. Написать программу поиска длиннейшей строки файла дублируя ее в файле.

f = **open**('11.txt'); L = f.**read().splitlines**(); f.**close**(); p=0; ns=[]

**for** i **in** **range**(**len**(L)):

 **if** **len**(L[i])>p: p= **len**(L[i]); ns=[i] #список номеров строк максимальной длины

 **elif** **len**(L[i])==p: ns+=[i]

f = **open**('11.txt', 'w')

**for** i **in** range(len(L)): #дублируем ВСЕ строки максимальной длины

 f.**write**(x+'\n')

 **if** i **in** ns: f.**write**(x+'\n')

f.**close**()

1. Написать программу, которая копирует текст одного файла в другой два раза.

f = **open**('11.txt'); L = f.**readlines**(); f.**close**(); f = **open**('111.txt', 'w')

**if ord**(L[-1][-1])!=10: L[-1]+= '\n' #после последней строки нужен '\n'

f.**writelines**(L); f.**writelines**(L);f.**close**()

1. Написать программу соединяющую соответствующие строки двух файлов (если в каком-то файле строк больше, то оставшиеся без пары остаются неизменными).

f = **open**('11.txt'); L1 = f.**readlines**(); f.**close**()

**if ord**(L1[-1][-1])!=10: L1[-1]+= '\n'

f = **open**('12.txt'); L2 = f.**readlines**(); f.**close**(); f = **open**('13.txt', 'w')

**if ord**(L2[-1][-1])!=10: L2[-1]+= '\n'

**if** **len**(L1)<**len**(L2):

 **for** i **in** **range**(**len**(L1)): L2[i]=L1[i][:-1]+L2[i]#если L1[i] завершается '\n'-убираем

 f.**writelines**(L2)

**else**:

 **for** i **in** **range**(**len**(L2)): L1[i]=L2[i][:-1]+L1[i]

 f.**writelines**(L1)

f.**close**()

 Заметим, что элементы массивов L1 и L2 содержат в конце '\n', что учитывается.

*Тема 3.4. Работа с файлами.* ***Лекционное занятие (1 час)***

При работе с файловым объектом можно использовать ряд свойств и методов – большинство универсально и предполагают чтение/запись строк в указанной кодировке (str для текстовых файлов) или набора байт (bytes для двоичных файлов):

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| close() | Закрыть файл/ничего не делать, если уже закрыт. |
| detach() | Возвращает бинарный буфер от TextIOBase. |
| fileno() | Возвращает целочисленный дескриптор файла. |
| flush() | Сброс буфера потока в файл. |
| isatty() | True, если файловый поток интерактивный. |
| read(n) | Читает максимум n символов из файла. |
| readable() | Возвращает True, если файловый поток читаемый. |
| readline(n=-1) | Возвращает строку файла, читает менее n+1 байт. |
| readlines(n=-1) | Вернет список строк, читает менее n+1 байт/симв. |
| seek(offset,from=SEEK\_SET) | Изменяет позицию указателя чтения/записи. |
| seekable() | Возвращает True, если задан случайный доступ. |
| tell() | Возвращает текущую позицию курсора в файле. |
| truncate(size=None) | Усекает размер потока до size/текущей позиции. |
| writable() | True, если поток открыт для записи. |
| write(s) | Пишет s в файл, возвращает число символов. |
| writelines(lines) | Записывает список lines в файл. |

Рассмотрим примеры работы с некоторыми методами, не рассмотренными ранее.

Метод seek устанавливает позицию байта offset чтения/записи в файле.

Аргумент whence необязателен и по умолчанию 0. Доступные значения:

* 0 означает, что нужно сместить указатель на offset относительно начала файла.
* 1 означает, что нужно сместить указатель на offset относительно текущей позиции.
* 2 означает, что нужно сместить указатель на offset относительно конца файла.

Обратите внимание, что не все объекты файлов доступны для метода seek(). В текстовых файлах разрешены только запросы относительно начала файла, исключение составляет смещение указателя до самого конца файла с помощью txt\_file.seek(0, 2).

text = b'This is 1st line\nThis is 2nd line\nThis is 3rd line\n'

f = **open**('f11.txt', 'bw+'); f.**write**(text); f.**seek**(20, 0);s=f.**read**(10);**print**(s);

f.**seek**(10, 1);s=f.**read**(10);**print**(s); f.**seek**(-11, 2);s=f.**read**(10);**print**(s);f.**close**()

|  |  |
| --- | --- |
| **Рис.17.** Результат работы программы: |  |

Метод seekable() проверяет поддержку произвольного доступа.

|  |  |
| --- | --- |
| f = **open**('11.txt', 'rt')**if** (f.**seekable**()): **print**('доступ разрешен')**else**: **print**('доступ запрещен')f.**close**() | **Рис.18.** Результат работы программы |

Также рассмотрим пример работы с методом tell(), который возвращает текущую позицию указателя чтения/записи в файле в байтах. Установить указатель чтения/записи в файле в нужную позицию можно используя метод файла file.seek().

|  |  |
| --- | --- |
| text='This is 1st line\nThis is 2nd line\nThis is 3rd line\n'f=**open**('f11.txt','w+');f.**write**(text);k=f.**tell**();**print**(k);f.**close**() | **Рис.19**. Результат работы |

***3.4.Практическое занятие (1 час)***

1. Написать программу чтения случайной строки из файла.

**from** random **import** \*; f = **open**('11.txt'); s=f.**read**().**splitlines**(); **print**(**choice**(s))

Обратите внимание на **splitlines** – возвращает список строк без '\n'.

1. Написать программу чтения файла и вывода количества строк, букв латиницы и слов.

f=**open**('file1.txt'); res = f.**readlines**(); **print**('количество строк ', **len**(res))

f.**seek**(0); words = f.**read**().**split**(); k1=**len**(words); **print**('количество слов ',k1)

k2=**sum**(words, key=len); **print**('количество букв ',k2)

1. Написать программу, которая получает на вход файл, а затем нумерует строки в нем.

f=**open**('text.txt'); s=f.**read().splitlines**();f.**close**();f=**open**('text.txt','w')

**for** i **in** **range**(**len**(s)): f.**write**(**str**(i+1)+')'+s[i]+'\n')

f.**close**()

Обратите внимание на использование splitlines() для избавления от переносов строк.

1. Написать программу, которая получает на вход строку и выводит новую, заменяя все запрещенные слова звездочками \* (их количество равно числу букв в запретном слове). Запрещенные слова хранятся в файле forb.txt. Все слова в нем записаны в нижнем регистре. Программа должна заменить запрещенные слова, где бы они ни встречались, даже в середине другого слова. Замена производится независимо от регистра.

f=**open**('forb.txt'); forb =f.**read**().**split**(); f.**close**()#для теста forb.txt: нае гре мас ус

sold='Спустился Грека в глубокий погреб, Хотел погреться: наесться гречки! '

#верхняя строка – для тестирования. Реально: sold=**input**('Введи исходную строку:)'

**print**(sold); s=sold #вывод строки до правки

**for** x **in** forb: s=s.**lower**().**replace**(x,'\*'\***len**(x))#после замены все буквы строчные

**for** i **in** **range**(**len**(s)):

 **if** s[i]!='\*' **and** sold[i].**isupper**():#где нет замены восстанавливаем заглавные

 s=s[:i]+sold[i]+s[i+1:]

**print**(s)

*Тема 3.5. Работа с файлами.* ***Лекционное занятие (1 час)***

В последнее время развитие IT переместилось от алгоритмов к работе с большими объемами разнородных данных, разработке универсальных форматов их хранения. Назовем наиболее популярные из них:

* CSV (Comma-Separated Values – значения, разделенные запятыми);
* JSON (JavaScript Object Notation, 1999)–текстовый формат, основанный на JavaScript;
* XML (eXtensible Markup Language – расширяемый язык разметки);
* YAML (YAML Ain’t Markup Language – «YAML - Не язык разметки»);
* INI (Initialization file - файл инициализации); и др.

Большинство этих форматов доступны Python через различнные модули и пакеты.

Остановимся на JSON файлах – по сути межъязыково стандарте данных. Файлы JSON читают/пишут все Си-подобные языки, а также они легко читается людьми:

|  |  |
| --- | --- |
| **Рис. 20.** Вид json-файла |  |

JSON-текст представляет собой одну из двух структур:

* набор пар ключ: значение (словарь), где ключ – строка, а значение – любой тип;
* упорядоченный набор значений (список).

|  |  |
| --- | --- |
| Значением может являться:* строка (в кавычках);
* число;
 | * логическое значение (true/false);
* null;
* одна из структур.
 |

В стандартную комплектацию Python входит модуль [**json**](https://docs.python.org/3/library/json.html) для кодирования и декодирования JSON–данных, подключаемый следующим образом:

**import** json

Для чтения в модуле json есть два метода:

* json.**load** – считывает файл в формате JSON и возвращает объекты Python;
* json.**loads** – считывает строку в формате JSON и возвращает объекты Python.

Пример чтения данных файла в формате JSON в объект Python:

**import** json

**with** **open**('data\_file.json') **as** f: templates = json.**load**(f)

**print**(templates)



**Рис.21.** Результат работы программы

Пример считывания строки в формате JSON в объект Python:

**import** json

**with** **open**('sw\_tmp.json') **as** f: file\_content = f.**read**(); templates = json.**loads**(file\_content)

Подобно сериализации, здесь также есть таблица преобразования типов:

|  |  |
| --- | --- |
| **JSON** | **Python** |
| object | dict |
| array | list |
| string | str |
| number (int) | int |
| number (real) | float |
| true | True |
| false | False |
| null | None |

Для записи информации в формате JSON в модуле json есть два метода:

* json.**dump** – запись объекта Python в файл в формате JSON;
* json.**dumps** – чтение строки в формате JSON.

Пример преобразования объекта в строку в формате JSON:

**import** json

data = {

 "student": {

 "name": "Ivan Efremov",

 "species": "Computer Science"

 }

}

**with** **open**("data\_file.json", "w") **as** write\_file: json.**dump**(data, write\_file)

В результате работы программы будет создан следующий файл:



**Рис. 22.** Вид файла

Функцию json.**dump**() используют для сериализации объекта Python в файл JSON, а json.**dumps**() – для преобразования данных JSON в строки для анализа и печати.

Оба метода dump() и dumps() используют одни и те же именованные аргументы. Первый – количество пробелов в отступе. Можно использовать именованный аргумент indent, для указания размера отступа *во вложенных структурах*.

Второй – separators: по умолчанию – двойной кортеж строк разделителя («, «, «: «), но обычно в качестве альтернативы для компактного JSON используют («,», «:»). Пример:

**import** json

person = '{"Name": "Andrew","City":"Russian", "Number":90014, "Age": 28,"Subject": ["Data Structure","Computer Graphics", "Discrete mathematics"]}'

per\_dict = json.**loads**(person); **print**(json.**dumps**(per\_dict, indent = 3, sort\_keys= True))

В приведенном выше коде мы предоставили 5 пробелов для аргумента отступа, а ключи отсортированы в порядке возрастания.

|  |  |
| --- | --- |
| **Рис.23.** Результат работы программы: |  |

Изменим значение отступа:

**import** json

person = '{"Name": "Andrew","City":"Russian", "Number":90014, "Age": 28,"Subject": ["Data Structure","Computer Graphics", "Discrete mathematics"]}'

per\_dict = json.**loads**(person); **print**(json.**dumps**(per\_dict, indent = 5, sort\_keys= True))

|  |  |
| --- | --- |
| **Рис.24.** Результат работы программы: |  |

***3.5.Практическое занятие (1 час)***

1. Написать программу подсчета строк файла, где 1-й и последний символы совпадают.

f=**open**('111.txt'); k=0

**for** x **in** f:

 x=x.**strip**()

 **if** x[0]==x[-1]: k+=1

**print**(k)

Обратите внимание, что возможно решение без занесения элементов файла в список. Также применен метод strip() для того чтобы отделить возможный знак переноса строки, который может быть считан вместе с текущей строкой.

1. Написать программу, которая заменит в файле все строки 101 на 000.

f=**open**('111.txt'); s = f.**readlines**(); f.**close**()

**for** i **in** **range**(**len**(s)):

 **if** s[i].**strip**()=='101': s[i]='000'+'\n'

f=**open**('111.txt','w'); f.**writelines**(s); f.**close**()

Обратите внимание на использование метода strip.

1. Написать программу вычисления $y=\frac{2+sin^{2}x}{1+x^{2}}, x\in [a,b]$, шаг h, с выводом в файл.

**from** math **import** sin

f=**open**('101.txt','w'); a,b,h=1,100,0.1; x=a

**while** x<=b: f.**write**(**str**((2+(**sin**(x))\*\*2)/(1+x\*\*2))+'\n'); x+=h

f.**close**()

1. Написать программу поиска в файле чисел отрезка [ -10000; 10000] максимального значения, кратного 3, и дописать его в конец файла.

f=**open**('101.txt', 'r+'); mx=-10001 #файл открыт для чтения и дописывания

**for** x **in** f:

 **if** **int**(x)%3==0 **and** **int**(x)>mx: mx=**int**(x)

f.**write**(**str**(mx)); f.**close**()

*Тема 3.6. Работа с файлами: примеры типовых задач.* ***Лекционное занятие (1 час)***

**Пример 1.** Обмен местами двух строк в файле.

Учтем, что в последней строке может отсутствовать '\n'. Позиции вводим с клавиатуры.

f = **open**('1.txt'); L = f.**readlines**();f.**close**();s = L[-1]; t= s[-1]

**if** (**len**(s)>0) **and** (t != '\n'): L[-1] += '\n' #для удобства унифицируем строки

p1 = **int**(**input**('Введи номер первой строки:'))

p2 = **int**(**input**('Введи номер второй строки:'))

**if** p2==p1 **or** **max**(p1, p2)>=**len**(L): **print**('ошибка ввода'); **exit**()

L[p1], L[p2]=L[p2], L[p1]

**if** t!= '\n': L[-1] = L[-1][:-1]#восстанавливаем значение последнего символа

f = **open**('1.txt', 'w'); f.**writelines**(L); f.**close**()

|  |  |  |
| --- | --- | --- |
| **Рис.25.** Вид файла до и после работы программы (пример 1): |  |  |

**Пример 2.** Реверсирование строк файла (перестановка в обратном порядке).

f = **open**('1.txt'); L = f.**readlines**(); f.**close**();s = L[-1]; t=s[-1]

**if** (**len**(s)>0)**and**((t != '\n')): L[-1] += '\n'

**for** i **in** **range**(**len**(L)//2+1: L[i], L[**len**(L)-i-1]= L[**len**(L)-i-1], L[i]

**if** t != '\n': L[-1] = L[-1][:-1] #восстанавливаем значение последнего символа

f = **open**('1.txt', 'w'); f.**writelines**(L); f.**close**()

|  |  |  |
| --- | --- | --- |
| **Рис.26.** Вид файла до и после работы программы (пример 2): |  |  |

**Пример 3.** Сортировка строк файла методом простого выбора (n-число строк).

Алгоритм данной сортировки списка x: в цикле для позиции начала от i=0 до n-2:

1. Найти позицию k минимума в списке на отрезке массива от i до n-1.
2. Поменять местами x[i] и x[k].

Эффективность алгоритма по скорости и памяти – O(n), где n-мощность списка.

f = **open**('1.txt'); L = f.**readlines**();f.**close**();s = L[-1]; t= s[-1]

**if** (**len**(s)>0)**and**((t != '\n')): L[-1] += '\n'

**for** i **in** **range**(**len**(L)-1):

 k = i

 **for** j **in** **range**(i+1,**len**(L)):

 **if** L[j]<L[k]: k=j

 L[i], L[k] = L[k], L[i]

**if** t!= '\n': L[-1] = L[-1][:-1]

f = **open**('1.txt', 'w'); f.**writelines**(L); f.**close**()

|  |  |  |
| --- | --- | --- |
| **Рис.27.** Вид файла до и после работы программы (пример 3): |  |  |

В этой задаче применим и стандартный алгоритм сортировки языка Python.

**Пример 4.** Объединение двух файлов в третий.

**with** **open**('1.txt') **as** f1, **open**('2.txt') **as** f2, **open**('3.txt', 'w') **as** f3:

L1 = f1.**readlines**(); L2 = f2.**readlines**()

**if** L1[-1][-1]!= '\n': L1[-1]+='\n'

f3.**writelines**(L1+L2)

|  |  |  |  |
| --- | --- | --- | --- |
| **Рис.28.** Вид исходны файлов и созданного после работы программы (пример 4): |  |  |  |

***3.6.Практическое занятие (1 час)***

1. Написать программу подсчета в текстовом файле количества русских гласных букв (любой регистр) и поиска символа чаще всего встречающегося после гласной буквы.

f= **open**('1.txt'); s=f.**read**(); f.**close**(); sn=[0]\*65536#счетчиков для всех символов

g='УЕЁЫАОЭЯИЮуеёыаоэяию'; k=0

**for** i **in** **range**(**len**(s)-1):

 **if** s[i] **in** g:

 k+=1; sn[**ord**(s[i+1])]+=1

**if** s[-1] **in** g: k+=1

**print**('Количество русских глачных в файле=',k); mx=**max**(sn)

**print**('Список символов чаще всего встречающихся в файле:')

**for** i **in** **range**(**len**(sn)):

 **if** sn[i]==mx: **print**(**chr**(i))

1. Написать программу поиска номеров строк, где больше всего букв 'A'.

f= **open**('1.txt'); s=f.**readlines**(); f.**close**(); L=[]; k=0; mx=s[0].**count**('A')

**if** mx>0: L+=[k]

**for** i **in** **range**(1, **len**(s)):

 m= s[i].**count**('A')

 **if** m>mx: mx=m; L=[i]

 **elif** m==mx: L+=[i]

**print**(\*L)

1. Написать программу поиска номеров строк, где больше всего букв ‘Z’, заменив в этих строках все символы Z их количеством и дописав эти строки в конец файла.

f= **open**('1.txt'); s=f.**readlines**(); f.**close**(); L=[]; k=0; mx=s[0].**count**('Z')

**if** mx>0: L+=[k]

**for** i **in** **range**(1, **len**(s)):

 m= s[i].**count**('Z')

 **if** m>mx: mx=m; L=[i]

 **elif** m==mx: L+=[i]

f= **open**('1.txt', 'a')

**for** i **in** **range**(**len**(L)): s[L[i]]=s[L[i]].**replace**('Z', **str**(mx)); f.**write**(s[L[i]])

f.**close**()

1. Написать программу, меняющую местами строки, где число букв 'N' максимально и строки, где она встречается ровно 1 раз.

Условие задачи некорректно. В самом деле: предположим мы имеем 5 строк, где буква 'N' встречается максимальное количество раз и 8 строк, где только однажды. Какую строку менять с какой, и что делать со строками, для которых нет пары? Примем решение менять строки в порядке их следования в файле, а «лишние» строки не менять.

f= **open**('1.txt'); s=f.**readlines**(); f.**close**(); L=[]; R=[]; k=0; mx=s[0].**count**('N')

**if** mx>0: L+=[k]

**if** mx==1:R+=[k]

**for** i **in** **range**(1, **len**(s)):

 m= s[i].**count**('N')

 **if** m==1: R+=[i]

 **if** m>mx: mx=m; L=[i]

 **elif** m==mx: L+=[i]

**if** mx<2: **print**('Ничего менять не надо'); **exit**()

**for** i **in** **range**(**min**(**len**(L), **len**(R))): s[L[i]], s[R[i]]= s[R[i]], s[L[i]]

f= **open**('1.txt''); f.**writelines**(s); f.**close**()

Здесь R, L – списки номеров строк, где 'N' входит 1 и максимум раз, соответственно.

|  |  |  |
| --- | --- | --- |
| **Рис.29.** Файл до и после работы программы (здесь для количества N максимум=3, тогда таких строк 3, а строк, где N входит 1 раз – 2, то есть 1↔2, 4↔5, строки 0 и 3 содержат букву N не 1 и не 3 раза, а строка 6 без пары): | 0sdfgNNsdf1NaNNsd2Nsdfg3Hfgg4NasdNNs5dfgNdsf6dsfNNfNs | 0sdfgNNsdf2Nsdfg1NaNNsd3Hfgg5dfgNdsf4NasdNNs6dsfNNfNs |

**Дополнительный материал к разделу 3.5**

 Ранее при обсуждении форматов организации баз данных было упомянуто, что популярные форматы типа CSV или JSON не обеспечивают защищенность данных в базе, а также требуют много места на носителе. Однако в разделе 3.2, при изучении бинарных файлов было показано, что возможна побайтная запись данных, которая затем доступны для чтения с использованием любого языка программирования, но лишь в том случае, если известен формат записи этих данных. Ниже представлены фрагменты программ для записи и чтения зарплатной ведомости, где база данных содержит поля «Фамилия», «Сумма к выдаче» и 3 поля даты выдачи:

Здесь фамилии грузятся из файла, а зарплата и дата выдачи генерируется случайными числами подходящих диапазонов. Вся информация пишется в бинарный файл блоками по 22 байта: фамилия – 15, зарплата – 3, дата: день – 1, месяц – 1 и год – 2.

**from** random **import** randint **as** rnd

fin=**open**('fam.txt'); f=**open**('fam1.dat','wb'); b=256;b2=b\*b

**for** t **in** fin:

 t=t.**strip**(); L=t+**chr**(0)\*(15-**len**(t))#дополняем нулями (15 символов на поле)

 zp=**rnd**(10000, 200000)#зарплата умещается в 3 байта (максимум - 16·106)

 d=**rnd**(1,28);m=**rnd**(1,12);y=**rnd**(2000,2025)#дата выдачи зарплаты 1б+1б+2б

 zp=[zp//b2,(zp%b2)//b,(zp%b2)%b]#зарплату -> в 3 байта

 dmy=[d,m,y//b,y%b]#дата выдачи в 4-х байтах

 f.**write**(**bytes**(L.**encode**('cp1251')))#пишем фамилию (по байту на символ cp1251)

 f.**write**(**bytes**(zp));f.**write**(**bytes**(dmy))#пишем деньги и дату (всего 22 байта)

L=**input**('Введи еще одну фамилию:');L=L.**strip**()+**chr**(0)\*(15-**len**(L));

f.**write**(**bytes**(L.**encode**('cp1251')))

zp=**int**(**input**('Введи зарплату:'));zp=[zp//b2,(zp%b2)//b,(zp%b2)%b]; f.**write**(**bytes**(zp))

d,m,y=**map**(**int**,**input**('Введи через пробелы цифрами день, месяц и год выдачи:').**split**())

dmy=[d,m,y//b,y%b]; f.**write**(**bytes**(dmy)); fin.**close**();f.**close**()

#Внимание!!! bytes без encode пишет только массивы байтов - чисел: каждое <256

Здесь последовательно из файла читаются нужные байты, где это необходимо, преобразуются в числа, а затем выводятся на экран в виде простой таблицы.

**print**('Фамилия'.center(14),'Зарплата'.center(7),' Выдано'.center(12))

f=**open**('fam1.dat','rb'); b=256; b2=b\*b #чтение, обработка, вывод; байт, и в квадрате

**try**:#контроль конца данных – по ошибке чтения

 **while** True:#бесконечный (до ошибки) цикл чтения

 L=f.**read**(15).**decode**('cp1251').**strip**(**chr**(0))#читаем,декодируем,чистим нуль-символы

 y=f.**read**(7);zp=y[0]\*b2+y[1]\*b+y[2]; d=y[3];m=y[4];y=y[5]\*b+y[6]#переводим из 256

 **print**('{:15}{:7d} {:02d} {:02d} {:4d}'.**format**(L,zp,d,m,y)) #форматный вывод

**except**: f.**close**()

Эти примеры можно использовать для собственных баз данных с большой скоростью доступа к ним: поскольку все пакеты одинакового размера, то есть прямой доступ к любой записи на чтение, удаление, добавление и изменение записи выполняется за **O**(1).

Попробуем еще больше сжать данные. Как видно из примера в таблице, это возможно!

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| 220-1=1048575 – предел для зарплаты | 25-1=31(max) | 24-1=15 | 211-1=2047 – предел для года |
| Зарплата (zp=257134) | d=27 | m=9 | y=2023 |

В этой 40-битной (**5-байтной**) цепочке для зарплаты, если она меньше указанного предела, достаточно 20 бит, а день, месяц и год потребуют по 5, 4 и 11 бит.

**Кодирование:** G=zp·220+d·215+m·211+y. B[0]=G//232; G= G%232; B[1]=G//224; G= G%224;

B[2]=G//216; G= G%216; B[3]=G//28; B[4]= G%28; f.**write**(**bytes**(B)) – запись 5 байтов.

**Чтение/декодирование:** g=f.**read**(5); G=g[0]·232+ g[1]·224+g[2]·216+g[3]·28+g[4].

zp=G//220;G=G%220; d=G//215;G=G%215; m=G//211; y=G%211. Строки читаются отдельно!

1. Особый случай – raw-файлы, которые здесь рассматриваться не будут. [↑](#footnote-ref-1)