МИНОБРНАУКИ РОССИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт математики, механики и компьютерных наук им. И. И. Воровича
Кафедра математического моделирования

«Проект 2 курс»

Мелехов Андрей Петрович

Задание 2. Файлы. Работа с файловой системой

Ростов-на-Дону

2021

При работе с физическими файлами на диске используются специальные логические файловые переменные («объекты файлы»). Встроенная функция open создает файловую переменную, которая обеспечивает связь с файлом, размещенным на диске компьютера.

После вызова функции open можно выполнять операции чтения и записи во внешний файл, используя методы полученного объекта. В конце работы файл закрывают.

Т. е. при работе с файлами нужно соблюдать последовательность операций:

1. **Открыть файл с помощью метода open( )**
2. **Чтение/запись данных из файла с помощью методов объектов файлов read( ) / write( )**
3. **Закрыть файл методом close( )**

Например, создаем файл и записываем в него две строки:

```
# (1) Создается новый файл для вывода:
f = open('data.txt', 'w')
# Файл будет создан в текущем рабочем каталоге.
# Параметр 'w' – файл открывается на запись.

# (2) Запись строк в файл:
f.write('Hello\n')
f.write('world\n')
# Метод write не записывает автоматически
# перенос строки. \n – записываем его сами.

# (3) Закрыть файл:
f.close( )
```

Открытие файла: функция open

Функция

f = open (file, mode = 'r', …)

открывает файл и возвращает файловую переменную.

file – путь к файлу. Путь файла может быть абсолютным (начинаться с буквы диска) или относительным (идти относительно расположения текущей рабочей папки).

mode – устанавливает режим открытия файла; по умолчанию используется значение 'r', что означает открытие для чтения в текстовом режиме. Другими распространенными значениями являются 'w' для записи (очистка содержимого файла, если он уже существовал) и 'a' для добавления.

Режимы открытия файла

Символ	Значение
'r'	открыть для чтения (по умолчанию)
'w'	открыть для записи (файл создается или очищается, если он существовал)
'x'	открытие на запись, если файл не существует, иначе исключение
'a'	открыть для записи в существующий файл (добавление в конец файла)
'b'	*бинарный режим ('rb', 'wb')*
't'	*текстовый режим (по умолчанию 'rt' = 'r')*
'+'	*открыть файл для обновления (чтение и запись) ('r+', 'w+')*

Примеры

Создать и открыть файл ’C:\temp.txt’ для записи:

```
output = open(r’C:\temp.txt’, ‘w’)  или
output = open(’C:\\temp.txt’, ‘w’)
```

Открыть существующий в текущей рабочей папке файл ‘data.txt’ для чтения:

```
input = open(‘data.txt’)            или
input = open(‘data.txt’, ‘r’)       или
input = open(‘data.txt’, ‘rt’)
```

Файловые методы	Описание
f.read( ) или f.read([n])	Читает весь файл в строку; или читает из файла до n байтов (символов).
f.readline( ) или f.readline([n])	Читает из файла очередную строку; читает одну строку, но не более n байт
f.readlines( )	Читает все строки и возвращает список
f.write(s)	Записывает строку s
f.writelines(lines)	Записывает все строки из lines
f.close( )	Закрывает файл
f.tell( )	Возвращает текущую позицию в файле
f.seek(offset , whence = 0)	Перемещает текущую позицию (относительно: whence = 0 – начала, 1 – текущей позиции, 2 – конца файла)

f.flush( )	**Выталкивает буферы вывода**
f.truncate([size])	**Усекает размер файла до текущей позиции (или до size байт)**
f.__next__( ) **next(f)**	**Итерации. Возвращает следующую строку или возбуждает исключение StopIteration.**

Пример 1. Создание файла

```
# Открывает файл (создает / очищает):
f = open('myfile.txt', 'w')
f.write('hello text file\n')     # Записывает строку текста
f.write('goodbye text file\n')
f.close( )          # Выталкивает данные из буфера на диск
```

Пример 2. Чтение данных из файла

```
# Открывает файл: чтение ‘r’ – по умолчанию:
f = open(‘myfile.txt’)
print(f.readline( ))        # Читает строку
print(f.readline( ))        # Читает строку
f.close( )
```

Выведет на экран:

```
hello text file
goodbye text file
```

1. **Если файл открыт в текстовом режиме, то читаются из него и записываются в него строки (или символы). Все остальные данные надо переводить в строки. Выполняется кодирование и декодирование символов. Используется кодировка, заданная по умолчанию. Можно выбрать другую.**
2. **Если файл открыт в бинарном режиме (двоичные файлы), то обмен идет в виде строк специального типа bytes.**
3. **Есть понятие курсора (текущей позиции в файле). При открытии файла курсор устанавливается в начало, в 0-ю позицию. При записи/чтении данных происходит смещение курсора. Называется последовательный доступ к файлу.**
4. **Работа с файлом идет через буфер. При закрытии файла методом close данные из буфера выталкиваются. При выходе из программы файл тоже закрывается. Метод flush.**

Текстовые файлы

Запись в текстовый файл. Чтобы открыть текстовый файл на запись, необходимо применить режим w (перезапись) или a (дозапись). Затем для записи применяется метод write(str), в который передается записываемая строка. Стоит отметить, что записывается именно строка, поэтому, если нужно записать числа, данные других типов, то их предварительно нужно конвертировать в строку.

Пример 3. Объекты языка Python должны записываться в текстовый файл только в виде строк:

```
S = 'Строка'                # строка
X, Y, Z = 43, 44, 45        # числа
D = {'a': 1, 'b': 2}        # словарь
```

```
L = [1, 2, 3]                          #   список
# Создает файл и открывает его на запись:
F = open(r‘D:\temp\datafile.txt’, ‘w’)
F.write(S + ‘\n’)                  # Запись строки и символа \n
F.write(‘%s,%s,%s\n’ % (X, Y, Z))      # числа в строки
F.write(str(L) + ‘\n’)             # Преобразует список в строку
F.write(str(D) + ‘\n’)             # Преобразует словарь в строку
F.close( )
```

Будет создан текстовый файл вида:

```
Строка
43,44,45
[1, 2, 3]
{'a': 1, 'b': 2}
```

Чтение из текстового файла. Для чтения файла он открывается с режимом r (Read), и затем мы можем считать его содержимое различными методами:

- **readline( ): считывает одну строку из файла**
- **read( ): считывает все содержимое файла в одну строку**
- **readlines( ): считывает все строки файла в список**

Считываются из текстового файла тоже строки. Интерпретатор Python автоматически не выполняет преобразование строк в числа или в объекты других типов.

Необходимо самостоятельно выполнить соответствующие преобразования!

Пример 4. Чтение из файла и перевод в нужный тип:

```
#  Открыть файл для чтения:
F = open(r‘D:\temp\datafile.txt’)
S = F.readline( )          # Прочитать одну строку  S=‘Строка\n’
S = S.rstrip( )            #  Удалить символ конца строки
```

Прочитаем целые числа:

```
line = F.readline( )  #  Читаем следующую строку из файла
                      #  line = ‘43,44,45\n’
XYZ = line.split(‘,’)     # Разбить на подстроки по запятым
                      #  XYZ = [‘43’, ’44’, ’45\n’]
numbers = [int(P) for P in XYZ]      #  Получить числа:
                      #  numbers = [43, 44, 45]
```

Чтобы преобразовать список и словарь можно воспользоваться встроенной функцией eval, которая интерпретирует строку как программный код на языке Python:

```
line = F.readline( )  #  Читаем следующую строку из файла
                      #  line = “[1, 2, 3]\n”
L = eval(line)        # Преобразовать строку в объект список
                      #  L = [1, 2, 3]
# Словарь:
line = F.readline( )  #  Читаем следующую строку из файла
                      #  line = “{‘a’: 1, ‘b’: 2}\n”
D = eval(line)        # Преобразовать строку в словарь
                      #  D = {‘a’: 1, ‘b’: 2}
```

Чтение всего файла

Создадим файл из трех строк:

```
f = open(r"file.txt", "r")
f.write('12345\n')      # запись 3 строк
f.write('abcde\n')
f.write('АБВГД')
print(f.name)           # печать имени файла
f.close()
```

Будет создан файл "file.txt" вида:

12345

abcde

АБВГД

Пример 5 (способ 1). Прочитать весь файл можно методом read:

```
f = open("file.txt", "r")
S = f.read( )  # весь файл записывается в строку S
print(S)
f.close()
```

Строка S будет иметь вид:

S = "12345\nabcde\nАБВГД"

При печати файл будет выведен в три строки (переносы):

12345

abcde

АБВГД

Пример 5 (способ 2). Прочитать весь файл также можно методом readlines:

```
f = open('file.txt')
list_str = f.readlines( )  # файл в список строк
```

Список строк будет иметь вид:

```
list_str = ['12345\n', 'abcde\n', 'АБВГД']
```

Пример 5 (способ 3). Читаем построчно пока не конец файла.

```
f = open("file.txt", "r")
line = f.readline()       # чтение строки
while line: # в конце файла – пустая строка
    print(line, end = '')
    line = f.readline() # чтение строки
close(f)
```

Пример 5 (способ 4). Удобней читать файл с помощью цикла for и итератора. В цикле for происходит построчное чтение всего файла:

```
for line in open('file.txt'):
    print(line, end = '')
```

Пример 5 (способ 5). Посимвольное чтение из файла

```
f = open("file.txt", "r")
char = f.read(1)        # чтение одного символа
while char: # в конце файла – пустая строка
    print(char, end = '')
    char = f.read(1)  # чтение одного символа
f.close()
```

Рабочий каталог

Если не указывать путь к файлу явно в функции open, то файлы будут сохраняться при записи, и программа будет искать при чтении их в этом каталоге.

Узнать текущий рабочий каталог можно так:

```
import os
cwd = os.getcwd() # возвращает текущий каталог
print(cwd)
```

Может вернуть не каталог, в котором находится программа!

В IDLE возвращает каталог программы. В Pyzo в меню Запуск → Change directory… или Запуск → Run file as script.

Или можно явно в программе указывать путь к файлу.

Использование кодировок

Посмотреть кодировку, используемую по умолчанию для работы с текстовыми файлами:

```
f = open("file.txt")
print(f.encoding)
```

Например, в Windows 10 вернул:

cp1251 (русская кодировка, в русскоязычной Windows)

или utf-8 (кодировка Unicode, в англоязычной Windows)

Посмотреть кодировку, установленную по умолчанию в Python (строки str имеют эту кодировку):

```
import sys
print(sys.getdefaultencoding())
```

Вывод:

```
utf-8
```

При записи строк str в файл будет происходить перевод из одной кодировки в другую. Не все символы из utf-8, есть в cp1251. Если не сможет перевести, выдаст ошибку!

Также если читать данные из файла, имеющего кодировку, отличную от заданной по умолчанию, будет неправильная интерпретация символов. Тоже может выдать ошибку.

Пример 6. Записываем символы в кодировке utf-8:

```
f = open("file.txt","w", encoding ='utf-8')
f.write('12345\n')
f.write('abcde\n')
f.write('АБВГД\300\u4E08') # символы À 丈
f.close()
```

Замечание. При попытке записать это в кодировке cp1251, выдал бы ошибку. В ней нет двух последних символов.

Создаст файл (кодировка utf-8):

```
12345
abcde
АБВГДÀ丈
```

Попытаемся прочитать этот файл (используя кодировку по умолчанию cp1251):

```
f = open(r"file.txt", "r")
S = f.read( )
print(S)
f.close()
```

Напечатает не то (т.к. коды символов в кодировках разные):

```
12345
abcde
РђР‘Р’Р“Р”ГЂдёЄ
```

Если использовать кодировку utf-8:

```
f = open(r"file.txt", "r", encoding='utf-8')
S = f.read( )
print(S)
f.close()
```

То выдаст правильный результат:

```
12345
abcde
АБВГДÀ丈
```

Пример 7. Записать данные в конец существующего файла

```
# Файл откроется
f = open(r"file.txt", "a")
# и курсор будет установлен в конец файла

# Записать строку в конец файла:
f.write('новая строка\n')
# или
# f.write('\nновая строка')
# если, в последней строке нет переноса '\n'
f.close()
```

Пример 8. **Создайте файл, содержащий случайное (не менее n1 и не более n2) количество случайных целых чисел из промежутка [–m, m]. Числа в файле разделены пробелами.**

```
import random
# random.seed()
n1 = 3                  # min-е количество
n2 = 10                 # max-е количество
m = 5                   # отрезок [-m, m]
n = random.randint(n1, n2)
f = open("numbers1.txt", "w")
for i in range(n):
    f.write('%d ' % random.randint(-m, m))
f.close()
```

Создаст файл вида:

```
4 -2 3 -5 3 -4 0 -1 -1
```

<u>Пример 9.</u> Чтение чисел из файла и запись их в список (числа в файле могут находиться в нескольких строках, разделены пробелами).

Например, файл “numbers2.txt” такого вида:

```
4    -2    3
   -5   3

-4 0 -1
-1
```

```
f = open("numbers2.txt")
L = []      # список для чисел из файла
for line in f: # читаем построчно
   # переводим в int и добавляем в список:
    L += map(int, line.split())
   # или
   # temp_list = [int(i) for i in line.split()]
   # L.extend(temp_list)
print(L)
f.close()
```

Печать списка:

[4, -2, 3, -5, 3, -4, 0, -1, -1]

Пример 10. Создать новый файл, в который поместить только четные числа из данного файла (каждое число в файлах расположено с новой строки).

```
f = open("numbers3.txt")          # из файла f
g = open("numbers4.txt", 'w')     # в файл g
for line in f:
    x = int(line)  # читаем очередное число
    # если оно четное, то его записываем:
    if x % 2 == 0: g.write(str(x) + '\n')
f.close()     # закрываем оба файла
g.close()
```

numbers3.txt:	numbers4.txt:
4	4
5	20
20 …	…

Прямой доступ к компонентам файла

f.tell( ) – возвращает текущую позицию в файле

f.seek(offset , whence = 0) – перемещает текущую позицию

<u>Пример 11.</u> Дан файл "data.txt" из двух строк:

Hello World!

Здравствуй мир!

```
f = open("data.txt")
for line in f:
    print(line, end='')
print('\n', '-' * 10)
f.seek(0)  # переместить курсор в начало
for line in f:
    print(line, end='')
f.close()
```

Выведет файл два раза:

Hello World!

Здравствуй мир!

Hello World!

Здравствуй мир!

Замечание 1. Если бы курсор не переместили в начало, второй раз файл бы не вывелся.

Замечание 2. Записать данные не даст в него, т. к. файл открыт только на чтение.

Замечание 3. С текстовыми файлами может работать некорректно. Лучше использовать с двоичными файлами.

Пример 12. Прямой доступ и открытие на чтение и запись

```
f = open("data.txt", 'r+') # чтение и запись
s = f.readline( )   # читаем первую строку
print(s)            # Печатает: Hello World!
print(f.tell())   # Выдает текущую позицию: 14
f.seek(0)  # Переместить курсор в начало файла
for i in range(12):
    s = f.read(1) # читаем по одному символу
    print(s, end = '') # и выводим их на экран

# Выведет:      Hello World!
print()
```

```
s = f.read(1)      # - читаем следующий символ
# заберет 2 символа с кодами 13 и 10!
print(ord(s),'=',ord('\n')) # выводим его код
# Выведет:  10 = 10
f.seek(17) # Устанавливаем в позицию 17:
#         Hello World!
#         Здравствуй мир!
# Записываем в нее букву 'A':
f.write('A')
f.close()
```

Изменит файл:

```
Hello World!
ЗдрАвствуй мир!
```

<u>Замечание.</u> Текстовые файлы в операционных системах (Windows, Unix, Mac OS) отличаются. Используются различные символы для обозначения конца строки: в Windows 2 символа с кодами 13, 10; и по 1 символу в Unix – код 10; и в Mac OS – код 13. В программе эта разница практически не заметна. При чтении/записи файлов в Python происходит конвертация: в программе всегда виден символ 10, а в файл записываются символы, в зависимости от того, в какой операционной системе Вы работаете.

Символы: '\n' и '\r':

```
ord('\n') == 10
ord('\r') == 13
```

Менеджер контекста with / as

Используется как альтернатива защищенному блоку try / finally.

Объекты файлов снабжены менеджером контекста, который автоматически закрывает файл после выполнения блока with независимо от того, было ли возбуждено исключение при его выполнении:

```
# Открывается файл внутри защищенного блока:
with open("test.txt") as f:
    # f – файловая переменная (объект файл)
    for line in f:
        print(line, end = '')
    # после завершения блока файл будет
    # закрыт даже, если была ошибка в блоке
```

Атрибуты объектов файлов

Объект, который возвращает функция open, содержит ссылку на существующий файл. Также в нем имеется информация об этом файле.

Свойство	Значение
closed	Состояние файла: True – файл закрыт, и False – файл открыт
encoding	Строка с названием кодировки файла
name	Имя файла
newlines	Содержит символы окончания строки, фактически используемые в файле. Может быть None, ‘\n’, ‘\r’, ‘\r\n’, или кортеж

	со всеми встреченными символами
mode	**Режим, в котором файл был открыт**
С помощью команды dir можно вывести все атрибуты файла: dir(f). Еще есть атрибуты: 'buffer', 'errors', 'fileno', 'isatty', 'readable', 'reconfigure', 'seekable', 'writable', 'write_through'.	

```
f = open(r"E:\temp\test.txt", "r")
s = f.read()  # если  этой строки нет,
# то f.newlines будет равно None
print(f.name) # "E:\temp\test.txt"
print(f.encoding) # cp1251
print([ord(ch) for ch in f.newlines])#[13, 10]
print(f.mode) # r
f.close()
```

Работа с файловой системой

https://docs.python.org/3/library/os.html

Модуль os в Python – это библиотека функций для работы с операционной системой. Включает методы для работы с файлами и каталогами.

Переименовать файл:

rename(<исходный файл>, <новое имя>)

Пример.

```
import os     # подключить модуль os
# установим текущий каталог:
os.chdir(r'E:\temp')
# в этом каталоге файл "1.txt" переименуем
# в файл "2.txt"
os.rename("1.txt", "2.txt")
```

Удалить файл:

remove(<имя файла>)

```
import os     # подключить модуль os
# можно указывать абсолютный путь:
os.remove(r'E:\temp\2.txt')
```

Проверить существование файла (или каталога):

https://docs.python.org/3/library/os.path.html

os.path.exists(<имя файла>)

```
if os.path.exists('1.txt'):
    print("Файл существует")
else:
    print("Файл не существует")
```

Узнать текущий рабочий каталог:

```
import os
print(os.getcwd()) # возвращает рабочий каталог
```

Установить текущий рабочий каталог:

```
os.chdir(r'D:\python\lab6')
```

mkdir(<каталог>) — **создает новую папку**

rmdir(<каталог>) — **удаляет папку**

listdir(<каталог>) — **показать содержимое папки**

Строки Unicode (кодировка utf-8)

UTF-8 – одна из кодировок Unicode (Unix), используется в Python для хранения строк str. В ней для хранения символов используется разное количество байт. Символы ASCII в этой кодировке занимают 1 байт (значения 0 – 127). Остальные символы от 2 до 3 байтов. Порядок кодирования приводится ниже:

Символ Юникода	Байт 0	Байт 1	Байт 2
U+0000 - U+007F	0nnnnnnn		
U+0080 - U+07FF	110nnnnn	10nnnnnn	
U+0800 - U+FFFF	1110nnnn	10nnnnnn	10nnnnnn

Перевод из Unicode в байты (тип bytes)

Посмотреть представление символов, можно выполнив преобразование символов во внутреннее представление (в тип bytes):

str.encode( ) или bytes(S, encoding) преобразуют строку в последовательность простых байтов и на основе объекта типа str создают объект типа bytes.

И, наоборот:

bytes.decode( ) или str(B, encoding) преобразуют последовательность простых байтов в строку и на основе объекта типа bytes создают объект типа str.

Пример. Для кодов ASCII (коды от 0 до 127)

```
s = chr(70) # берем число от 0 до 0x7f = 127
print(s)      # вернет 'F'
b = s.encode( ) # - перевод во внутреннее
# представление (bytes)
# или:
# b = bytes(s, encoding = 'utf8')
print(type(b), len(b), b[0])
```

Напечатает:

```
<class 'bytes'>    1      70
```

Тип bytes; занимает в памяти 1 байт; код 70.

Пример. Для кодов от 128 до 2047

```
s = chr(1070) # от 0x80 = 128 до 0x7ff = 2047
print(s)      # вернет 'Ю'
b = bytes(s, encoding = 'utf8')
# или:
# b = s.encode( )
print(len(b), b[0], b[1], b)
print(bin(b[0]),bin(b[1]))
```

Вернет:

```
2   208   174   b'\xd0\xae'
  0b11010000   0b10101110
```

2 байта с кодами 208 и 174

Пример. Для кодов от 2048 до 65535

```
# от 0x800 = 2048 до 0xffff = 65535
s = chr(20487)
print(s)          # вернет '婇'
b = bytes(s, encoding = 'utf8')
print(len(b), b[0], b[1], b[2], b)
print(bin(b[0]), bin(b[1]), bin(b[2]))
```

Вернет:

```
3 229 128 135 b'\xe5\x80\x87'
0b11100101 0b10000000 0b10000111
```

3 байта с кодами 229, 128 и 135

Двоичные файлы

Файл открывается в *двоичном режиме*, при добавлении в строку режима символа ‘b’ в функцию open. Содержимое *двоичных файлов* представляется в виде строк типа bytes, и оно передается программе без каких-либо изменений. Также тип bytes совместим с типом bytearray (изменяемый bytes).

```
bs = b'123abc' # можно создавать так b'…'
print(bs[0], bs[3]) # хранит числа: 49 97
print(bs)     # вывод: b'123abc'
s = bs.decode()      # перевод в строку
print(s[0], s[3])   # вывод: 1 a
print(s)      # вывод: 123abc
```

Откроем текстовый файл

12345

abcde

АБВГД

как двоичный (записан в кодировке ср1251):

Пример.

```
f = open(r"file.txt", "rb")
sb = f.read()
print(type(sb))  # тип <class 'bytes'>
print(sb)        # печатает sb
print(list(sb))  # печатаем по 1 байту
f.close()
```

Вернет:

b'12345\r\nabcde\r\n\xc0\xc1\xc2\xc3\xc4'

[49, 50, 51, 52, 53, 13, 10, 97, 98, 99, 100, 101, 13, 10, 192, 193, 194, 195, 196]

Если бы этот файл

12345

abcde

АБВГД

был записан в кодировке utf-8, то программа вернула бы:

b'12345\r\nabcde\r\n\xd0\x90\xd0\x91\xd0\x92\xd0\x93\xd0\x94'

[49, 50, 51, 52, 53, 13, 10, 97, 98, 99, 100, 101, 13, 10, 208, 144, 208, 145, 208, 146, 208, 147, 208, 148]

Хранение целых чисел в двоичных (битовых) файлах

Пример 1. Перевод целого числа int в bytes и обратно (int ↔ bytes)

Двоичные файлы хранят данные в виде строк bytes. Целое число можно перевести в bytes с помощью метода to_bytes:

int.to_bytes(length, byteorder, *, signed = False)

Параметры	**Описание**
length	количество байт выделяемое на число (вызывается исключение, если нельзя записать число в указанное число байт)
byteorder	определяет порядок байтов, используемый для представления целого числа ('big' или 'little')
signed	число со знаком (True) или нет (False)

Наоборот, данные из типа bytes в int можно перевести с помощью метода from_bytes:

int.from_bytes(bytes, byteorder, *, signed = False)

Параметры	**Описание**
bytes	данные типа bytes
byteorder	порядок байтов представления целого числа ('big' или 'little')
signed	число со знаком (True) или нет (False)

Родной порядок байт системы возвращает sys.byteorder.

```
import sys
# Родной порядок байт:
print(sys.byteorder)        # little
# Перевод числа 1024 в bytes (2 байта со знаком):
```

```
bx = int.to_bytes(1024, 2, byteorder = 'little',
signed = True)
print(bx)                    # b'\x00\x04'
# Обратный перевод числа 1024 из bytes (2 байта
со знаком):
x = int.from_bytes(bx, byteorder = 'little',
signed = True)
print(x)                     # 1024
```

Пример 2. Создание файла, содержащего случайное количество (от 3 до 10) случайных целых чисел из промежутка [–32768, 32767].

```
import random
random.seed()
n = random.randint(3, 10)
```

```
# Создаем/открываем битовый файл "numbers1.dat"
для записи:
f = open("numbers1.dat", "wb")
for i in range(n):
# Будем использовать формат числа 2 байта со знаком,
# Максимально допустимый диапазон: [-32768, 32767].
    x = random.randint(-32768, 32767)
    # Перевод в bytes:
    bx = x.to_bytes(2, byteorder = 'little',
                    signed = True)
    # Сохранение 2 байт в файл:
    f.write(bx)
f.close( )
```

Пример 3. Чтение целых чисел из битового файла и вывод их на экран

```
# Открываем битовый файл "numbers1.dat" для чтения:
f = open("numbers1.dat", "rb")
# читаем из него по 2 байта:
bx = f.read(2)
# Проверка на пустую строку bytes b'':
while bx != b'':
    # Перевод из bytes в число int:
    x = int.from_bytes(bx, byteorder='big',
                       signed=True)
    # выводим числа на экран:
    print(x)
    # читаем из файла по 2 байта:
    bx = f.read(2)
f.close()
```

Модуль Pickle (сериализация объектов Python)

https://docs.python.org/3/library/pickle.html#module-pickle

Модуль pickle реализует двоичные протоколы для сериализации и десериализации структуры объектов Python. Преобразует объекты в поток байтов (упаковка), и, наоборот, поток байтов (из двоичного файла или байтоподобного объекта) преобразуется обратно в иерархию объектов (распаковка).

Замечание. Небезопасен, как и eval или exec. Рекомендуется распаковывать только файлы, полученные из надежного источника.

Модуль pickle позволяет сохранять в файлах практически любые объекты Python без необходимости с нашей стороны выполнять какие-либо преобразования.

Например, сохранить данные в файле:

```
import pickle  # подключаем модуль pickle
F = open('datafile.pkl', 'wb') # открываем файл
# Модуль pickle запишет в файл любой объект:
pickle.dump(10, F)          # Записываем число
pickle.dump([1, 2, 3], F)  # Список
D = {'a': 1, 'b': 2}
pickle.dump(D, F)           # Словарь
F.close( )
```

Теперь откроем файл и прочитаем из него данные:

```
import pickle  # подключаем модуль pickle
F = open('datafile.pkl', 'rb')
x = pickle.load(F)  # Загружает число в x
L = pickle.load(F)  # Загружает список
D = pickle.load(F)  # Загружает словарь
print(x, type(x))  # Вернет: 10 <class 'int'>
print(L, type(L))  # [1, 2, 3] <class 'list'>
print(D, type(D))  # {'a': 1, 'b': 2}
                   #  <class 'dict'>
F.close( )
```

Нам не нужно ни переводить данные в строки при записи, ни расшифровывать их при чтении. Модуль pickle делает это сам.

Загрузить данные из файла:

```
F = open('datafile.pkl', 'rb')
s = F.read()
print(type(s))    # вернет <class 'bytes'>
                  # двоичный файл
print(s) # вернет:
```

b'\x80\x03K\n.\x80\x03]q\x00(K\x01K\x02K\x03e.\x80\x03}q\x00(
X\x01\x00\x00\x00aq\x01K\x01X\x01\x00\x00\x00bq\x02K\x02u.'

Упаковывает данные (pickle, консервирует).

Можно сохранять набор объектов одного типа, например, набор чисел:

```
import pickle
n = 5
# Записываем n чисел:
f = open('datafile1.pkl', 'wb')
for i in range(n):
    pickle.dump(i, f)
f.close( )
```

```
# Читаем n чисел:
f = open('datafile1.pkl', 'rb')
L = []
for i in range(n):
    L.append(pickle.load(f))
print(L) #  [0, 1, 2, 3, 4]
f.close( )
```

Замечание. **Если мы не знаем, сколько чисел в файле, и попытаемся прочитать лишний элемент, выдаст ошибку: EOFError: Ran out of input.**

Можно использовать защищенный блок для чтения файлов с произвольным количеством элементов.

Пример. **Прочитать все числа из файла**

```
import pickle
f = open('datafile1.pkl', 'rb')
L = []
# В цикле:
while True:
    try: # Пытаемся прочитать из файла:
        L.append(pickle.load(f))
    except EOFError: # Если не получилось
        break    # выходим из цикла
print(L)     # Печать:  [0, 1, 2, 3, 4]
f.close( )
```

Кроме методов записи / чтения в файл (dump / load), есть методы просто упаковки (dumps / loads):

```
import pickle
t1 = [1, 2, 3]
s = pickle.dumps(t1) # упаковали
print(type(s))       # тип s: <class 'bytes'>
t2 = pickle.loads(s) # распаковали
print(t2)            # [1, 2, 3]
```

II. Задания

Замечание. Удобно работать с файлами, находящимися в одном каталоге с программой. Тогда не нужно указывать полный путь к файлу, достаточно только имя файла. Для этого нужно, чтобы каталог программы был текущим рабочим каталогом. В IDLE они совпадают. В Pyzo по умолчанию нет. Но можно настроить в меню Pyzo: “Run (Запуск) → Change directory when executing file”. Или запускать программу с помощью следующей команды меню: Запуск → Run file as script.

Работа с числами в текстовых файлах

1. Создать файл, содержащий целые числа. В файле создается случайное количество строк (от 1 до 5), в каждой строке случайное количество целых чисел (от 0 до 5). Числа в каждой строке, разделены пробелами.
2. Создать файл, содержащий все двузначные натуральные числа, делящиеся на заданное число m (каждое число в новой строке).

3. Создайте файл, содержащий случайное (не менее $n1$ и не более $n2$) количество случайных целых чисел из промежутка $[-m, m]$. Числа в файле разделены пробелами.

Создайте текстовый файл, содержащий целые числа, разделенные пробелами в одной строке (с помощью приложения Блокнот или WordPad).

4. Читать числа из файла и вывести их на экран каждое с новой строки.
5. Найти в файле максимальное число и вывести его на экран.
6. Подсчитать произведение положительных чисел из файла.
7. Чтение чисел из файла и вычисление суммы двузначных чисел.
8. Проверить, есть ли в файле числа, оканчивающиеся цифрой 3.

Создайте текстовый файл с целыми числами, находящимися в нескольких строках файла, разделенные пробелами внутри строк (с помощью приложения Блокнот или WordPad).

9. Чтение чисел из существующего файла и создание двух новых файлов: в один записываются четные числа, в другой – нечетные (каждое число с новой строки).
10. Чтение чисел из существующего файла и создание нового файла, в который записываются те числа, в десятичном представлении которых использованы только нечетные цифры (числа разделены пробелами в одной строке).
11. Создать два новых файла: в первый поместить положительные компоненты файла, а во второй все остальные (числа разделены пробелами в одной строке).
12. Создать новый файл, в который поместить только четные числа из файла (каждое число остается в строке с тем же номером; если в строке не было четных чисел, то в новом файле эта строка пустая).

Чтение и запись текстовых файлов в разных кодировках

13. Создайте произвольный текстовый файл “text_encoding_1.txt” (с помощью приложения Блокнот или WordPad). Используйте в этом файле как латинские, так и кириллические буквы.

14. Посмотрите кодировку, установленную по умолчанию для работы с текстовыми файлами (f.encoding). Прочитайте созданный файл, используя разные кодировки (utf-8, cp1251 или другие), и выведите его содержимое на экран.

    Замечание. Для проверки в программу добавьте комментарий, например, “кодировка по умолчанию …, корректно все данные прочитались в кодировке …, в другой кодировке …”.

15. Создайте файл, содержащий разные символы Unicode (например, английские, русские и иероглифы). Попробуйте сохранить его, используя разные кодировки (в файлы “text_encoding_utf8.txt” и т. п.). Откройте созданные файлы в Блокноте.

Замечание. Для проверки в программу добавьте комментарий, например, “удалось сохранить в кодировке …, корректно отображаются в Блокноте …”.

Работа с файловой системой

Напишите программы, которые выполняют следующие действия

16. Создайте новый каталог. Создайте в этом каталоге файл.
17. Переименуйте существующий файл. Проверьте существование файла.
18. Удалите существующий файл. Проверьте существование файла.
19. Покажите содержимое текущей папки (вывести на экран имена файлов).

Хранение целых чисел в двоичных файлах

В примерах (см. стр. 50–54) демонстрируются переводы целого числа int в тип bytes и обратно; создание битового файла, содержащего целые числа и чтение битового файла и вывод его содержимого на экран.

20. Создать файл, содержащий простые числа из промежутка [100, 200]. Проверку, является ли натуральное число простым, оформить в виде функции.

21. Создать файл, содержащий случайное количество (от 10 до 15) случайных натуральных чисел.

22. Из файла целых чисел создать новый файл по правилу: все подряд идущие значения одного знака суммируются, группа нулей заменяется одним нулем.

23. Из двух файлов целых чисел, упорядоченных по возрастанию, создать новый файл чисел, упорядоченных по возрастанию.

Можно использовать прямой доступ к файлу (методы seek и tell)

24. Найти в файле максимальное и минимальное число и поменять их местами.

25. Найти в файле максимальное число и удалить его из файла.

Замечание. Алгоритм удаления: в компоненту, содержащую максимальное число, скопировать последнюю компоненту файла, а затем удалить последнюю компоненту с помощью truncate.

26. Удалить из файла наименьшее нечетное число (см. замечание к предыдущей задаче).

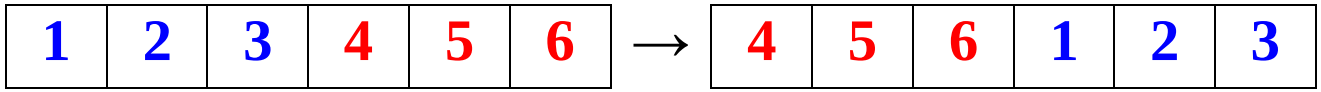
27. Создать еще один файл, в который поместить числа из файла в обратном порядке.

28. Изменить в файле порядок следования элементов на обратный, меняя местами равноудаленные от концов элементы файла.

29. Проверить, есть ли в файле простые числа, вывести на экран сообщение.

30. Поменять местами в файле вторую и предпоследнюю компоненты.

31. Поменять местами компоненты файла по схеме:



(если файл содержит нечетное количество компонент, то предварительно удалить последнюю).

Модуль Pickle

32. Сохранение произвольный список (list) в файл с помощью модуля Pickle. Прочитайте обратно его из этого файла и выведите на экран.

33. Сохранение произвольный словарь (dict) в файл с помощью модуля Pickle. Прочитайте обратно его из этого файла и выведите на экран.

34. Сохранение произвольное множество (set) в файл с помощью модуля Pickle. Прочитайте обратно его из этого файла и выведите на экран.