Работа с файлами

Для работы с файлами можно, как в Сях, открывать файл, работать с ним и потом закрывать. Но мы не программисты, а питонисты, поэтому думать о том, чст что-то там нужно закрывать это для лохов.
Специально для нас в питончике есть Контекстный менеджер ***with***. Он может работать с любым объектом, в котором реализованы методы ***__enter__*** и ***__exit__***. Порядок такой: отрабатываем вход, присваиваем в рамках локального куска кода объекту новое локальное имя открытого объекта; работаем с ним, по выходе из блока автоматически срабатывает выход

```
>>> class CM:
...     def __init__(self, val):
...         self.val = val
...     def __enter__(self):
...         print("> Enter")
...     def __exit__(self, *args):
...         print("> Exit:", args)
...
>>> with CM(100500) as context:
...     print("Working")
...     raise SyntaxError
...     print("Not Working")
...
> Enter
Working
> Exit: (<class 'SyntaxError'>, SyntaxError(), <traceback object at 0x7f1d3b6f65c0>)
Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
SyntaxError: None

>>> class CM:
...     def __init__(self, val):
...         self.val = val
...     def __enter__(self):
...         print("> Enter")
...     def __exit__(self, *args):
...         print("> Exit:", args)
...
>>> with CM(100500) as context:
...     print("Working")
...     # raise SyntaxError
...     print("Not Working")
...
> Enter
Working
Not Working
> Exit: (None, None, None)
```

Такая try-finally моделька получилась

```
>>> class CM:
...     def __init__(self, val):
...         self.val = val
...     def __enter__(self):
...         print("> Enter")
...     def __exit__(self, *args):
...         print("> Exit:", args)
...         return self.val
...
>>>
>>> with CM(100500) as context:
...     print("Working")
...     raise SyntaxError
...     print("Not Working")
...
> Enter
Working
> Exit: (<class 'SyntaxError'>, SyntaxError(), <traceback object at 0x7f1d3b6f71c0>)
```

файл в питоне - как раз таки попадает под работу с контекстным менеджером

Теперь про буковки, с которыми нам сталкиваться:

Тип Байты - тоже как бы строки, но только АСКИИ

```
>>> b"qwertyhj"
b'qwertyhj'
>>> b = b'qwertyui'
>>> dir(b)
['__add__', '__bytes__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__',
'__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', 'capitalize', 'center', 'count', 'decode', 'endswith',
'expandtabs', 'find', 'fromhex', 'hex', 'index', 'isalnum', 'isalpha', 'isascii', 'isdigit',
'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans',
'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition',
'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate',
'upper', 'zfill']
>>> type(b)
<class 'bytes'>
>>> b[1]
119
>>> chr(b[1])
'w'
```

Bytes хранит почти СИшные строки

Следующий тип - изменяемые Байты - почти дин.массив - bytearray

```
>>> b = bytearray((1, 2, 3, 4, 5, 6))
>>> b
bytearray(b'\x01\x02\x03\x04\x05\x06')
>>> b = bytearray((111, 112, 113, 114, 115, 116))
>>> b
bytearray(b'opqrst')
>>> dir(b)
['__add__', '__alloc__', '__class__', '__contains__', '__delattr__', '__delitem__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__getstate__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__',
'__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append',
'capitalize', 'center', 'clear', 'copy', 'count', 'decode', 'endswith', 'expandtabs', 'extend',
'find', 'fromhex', 'hex', 'index', 'insert', 'isalnum', 'isalpha', 'isascii', 'isdigit', 'islower',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'pop',
'remove', 'removeprefix', 'removesuffix', 'replace', 'reverse', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
>>> b[2]
113
>>> b[3] = 121
>>> b
bytearray(b'opqyst')
```

Такой массивчик можно модифицировать, однако все значения больше-равные 128 печатаются не буковками, а Хексом (как мы увидели, некоторые меньше тоже печатаются (там где не определены символы алфавита английского или где спецсимволы))

Возникает вопрос: Можно ли Питоновскую Строку в Байты перегнать?
Байты 0-127 - один в один по АСКИИ. А с остальными что?

>>> txt = "Вопрос"
>>> ord(txt[0])
1042
>>> hex(ord(txt[0]))
'0x412'

Получили двухбайтовые или больше индексы
Это происходит потому, что в кодировках 0-127 фиксированные символы, дальше - какие-то символы, зависит от кодировки, с каким она языком итд. Отсюда происходит очень смешная штука. которую в программистской среде прозвали БНОПНЯ

На разных устройствах могут быть разные кодировки, соответственно при переводе одного слова из кодировки в байты, а оттуда в другую кодировку может получиться всякая смешная дичь)
Собственно, “Бнопня” это то, что выдавалось, когда кто-то с Линуха писал в файл слово “Вопрос”, сохранял в своей старой виндусовой кодировке (WINDOWS-1251), а потом этот файл читали с кодировкой KOI8-R

Собственно, давайте глазками посмотрим на такие перегоны. А помогут нам методы перевода из строки в байты (***encode***) и из байт в строку (***decode***)

>>> txt.encode("KOI8-R")
b'\xf7\xcf\xd0\xd2\xcf\xd3'
>>>
>>> txt.encode("CP866")
b'\x82\xae\xaf\xe0\xae\xe1'
>>> txt.encode("WINDOWS-1251")
b'\xc2\xee\xef\xf0\xee\xf1'
>>> txt.encode("WINDOWS-1251").decode('KOI8-R')
'бНОПНЯ'
>>> 'Внимание'.encode("WINDOWS-1251").decode('KOI8-R')
'бМХЛЮМХЕ'
>>> 'Внимание'.encode("UTF-8")
b'\xd0\x92\xd0\xbd\xd0\xb8\xd0\xbc\xd0\xb0\xd0\xbd\xd0\xb8\xd0\xb5'
>>> 'Внимание!'.encode("UTF-8")
b'\xd0\x92\xd0\xbd\xd0\xb8\xd0\xbc\xd0\xb0\xd0\xbd\xd0\xb8\xd0\xb5!'
>>> 'Внимание!'.encode("UTF-8").decode("WINDOWS-1251")
'Р’PSPёPjP°PSPёPµ!'

Немножко Просто Файлов: возьмём вот такой файлик

stephen@The-Night-Road:~$ cat tat
READDIR(3) Linux Programmer's Manual READDIR(3)

NAME
readdir - read a directory

… <тут реааааально много текста)))>

COLOPHON
This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.

2019-03-06 READDIR(3)
stephen@The-Night-Road:~$ python3.11
Python 3.11.0rc1 (main, Aug 12 2022, 10:02:14) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.

Почитаем его:
метод ***read*** читает весь текст целиком

>>> with open("tat") as f:
... print(f.read())
...
READDIR(3) Linux Programmer's Manual READDIR(3)

NAME
readdir - read a directory

...

COLOPHON
This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.

2019-03-06 READDIR(3)

По файлу можно построчно итерироваться

```
>>> with open("tat") as f:
...     i = 0
...     for line in f:
...             print(line)
...             i += 1
...             if i > 5:
...                     break
...
READDIR(3)              Linux Programmer's Manual              READDIR(3)
NAME
       readdir - read a directory

SYNOPSIS
```

В Питончике мы также можем непосредственно взаимодействовать с потоками ввода-вывода. А, как мы помним с ОСей, тк поток в файл или в поток ввода-вывода ничем действия не отличаются, можно переприсваивать одно другому итд, как мы делали через ***dup*** в ОСях

```
>>> import sys
>>> for l in sys.stdin:
...     print(repr(l))
...
qwertyu
'qwertyu\n'
asdfghjytredfnj
'asdfghjytredfnj\n'
cvbnjhrdvbnjuyt
'cvbnjhrdvbnjuyt\n'

stephen@The-Night-Road:~$ cat > test.txt
qwerthgfdfyu
sdfghjytrdxbn
123456789
 nmjytrewsdfghjki
hello
wertgfx
stephen@The-Night-Road:~$ python3.11
Python 3.11.0rc1 (main, Aug 12 2022, 10:02:14) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> import sys
>>>
>>> with open('test.txt') as f:
...     sys.stdin = f
...     print(repr(input()))
...
'qwerthgfdfyu' # вот эта штука сама напечаталась из файла передавшись на вход будто бы
```

```
stephen@The-Night-Road:~$ cal > cal # записали в файл календарик
stephen@The-Night-Road:~$ python3.11
Python 3.11.0rc1 (main, Aug 12 2022, 10:02:14) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> with open('cal') as f:
...     for l in f:
...             print(l)
...
   Ноябрь 2023
Вс Пн Вт Ср Чт Пт Сб
                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



>>> with open('cal', 'rb') as f:
...     for l in f:
...             print(l)
...
b'    \xd0\x9d\xd0\xbe\xd1\x8f\xd0\xb1\xd1\x80\xd1\x8c 2023       \n'
b'\xd0\x92\xd1\x81 \xd0\x9f\xd0\xbd \xd0\x92\xd1\x82 \xd0\xa1\xd1\x80
\xd0\xa7\xd1\x82 \xd0\x9f\xd1\x82 \xd0\xa1\xd0\xb1  \n'
b'          1  2  3  4  \n'
b' 5  6  7  8  9 10 11  \n'
b'12 13 14 15 16 17 18  \n'
b'19 20 21 22 23 24 25  \n'
b'26 27 28 29 30        \n'
b'                      \n'

stephen@The-Night-Road:~$ hexdump -C cal
00000000  20 20 20 20 d0 9d d0 be  d1 8f d0 b1 d1 80 d1 8c  |    ............|
00000010  20 32 30 32 33 20 20 20  20 20 20 20 0a d0 92 d1  | 2023       ....|
00000020  81 20 d0 9f d0 bd 20 d0  92 d1 82 20 d0 a1 d1 80  |. .... .... ....|
00000030  20 d0 a7 d1 82 20 d0 9f  d1 82 20 d0 a1 d0 b1 20  | .... .... .... |
00000040  20 0a 20 20 20 20 20 20  20 20 20 20 31 20 20 32  | .          1  2|
00000050  20 20 33 20 20 34 20 20  0a 20 35 20 20 36 20 20  |  3  4  . 5  6  |
00000060  37 20 20 38 20 20 39 20  31 30 20 31 31 20 20 0a  |7  8  9 10 11  .|
00000070  31 32 20 31 33 20 31 34  20 31 35 20 31 36 20 31  |12 13 14 15 16 1|
00000080  37 20 31 38 20 20 0a 31  39 20 32 30 20 32 31 20  |7 18  .19 20 21 |
00000090  32 32 20 32 33 20 32 34  20 32 35 20 20 0a 32 36  |22 23 24 25  .26|
000000a0  20 32 37 20 32 38 20 32  39 20 33 30 20 20 20 20  | 27 28 29 30    |
000000b0  20 20 20 20 0a 20 20 20  20 20 20 20 20 20 20 20  |    .           |
000000c0  20 20 20 20 20 20 20 20  20 20 20 20 0a           |          .|
000000cc
```

```
stephen@The-Night-Road:~$ python3.11
Python 3.11.0rc1 (main, Aug 12 2022, 10:02:14) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> with open('cal') as f:
...     for l in f:
...             print(*(hex(ord(c)) for c in l))
...
0x20 0x20 0x20 0x20 0x41d 0x43e 0x44f 0x431 0x440 0x44c 0x20 0x32 0x30 0x32 0x33
0x20 0x20 0x20 0x20 0x20 0x20 0x20 0xa
0x412 0x441 0x20 0x41f 0x43d 0x20 0x412 0x442 0x20 0x421 0x440 0x20 0x427 0x442
0x20 0x41f 0x442 0x20 0x421 0x431 0x20 0x20 0xa
0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x31 0x20 0x20 0x32 0x20 0x20
0x33 0x20 0x20 0x34 0x20 0x20 0xa
0x20 0x35 0x20 0x20 0x36 0x20 0x20 0x37 0x20 0x20 0x38 0x20 0x20 0x39 0x20 0x31
0x30 0x20 0x31 0x31 0x20 0x20 0xa
0x31 0x32 0x20 0x31 0x33 0x20 0x31 0x34 0x20 0x31 0x35 0x20 0x31 0x36 0x20 0x31
0x37 0x20 0x31 0x38 0x20 0x20 0xa
0x31 0x39 0x20 0x32 0x30 0x20 0x32 0x31 0x20 0x32 0x32 0x20 0x32 0x33 0x20 0x32
0x34 0x20 0x32 0x35 0x20 0x20 0xa
0x32 0x36 0x20 0x32 0x37 0x20 0x32 0x38 0x20 0x32 0x39 0x20 0x33 0x30 0x20 0x20
0x20 0x20 0x20 0x20 0x20 0x20 0x20 0xa
0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20
0x20 0x20 0x20 0x20 0x20 0x20 0xa


>>> with open('cal') as f:
...     line = f.readline()
...     print(line)
...
   Ноябрь 2023

>>> with open('cal', 'rb') as f:
...     line = f.readline()
...     print(line)
...
b'    \xd0\x9d\xd0\xbe\xd1\x8f\xd0\xb1\xd1\x80\xd1\x8c 2023       \n'

>>> with open('cal', 'rb') as f:
...     line = f.readline()
...     print(line)
...     print(line.decode("UTF8"))
...
b'    \xd0\x9d\xd0\xbe\xd1\x8f\xd0\xb1\xd1\x80\xd1\x8c 2023       \n'
   Ноябрь 2023
```

```
>>> import locale
>>> locale.getdefaultlocale() # поставили в системе кодировку по умолчанию на всякий случай
>>> with open('cal', 'rb') as f:
...     line = f.readline()
...     print(line)
...     print(line.decode())
...
b'    \xd0\x9d\xd0\xbe\xd1\x8f\xd0\xb1\xd1\x80\xd1\x8c 2023       \n'
    Ноябрь 2023

>>> with open('cal', 'rb') as f:
...     line = f.readline()
...     print(line)
...     print(line.decode("iso2022_jp"))
...
b'    \xd0\x9d\xd0\xbe\xd1\x8f\xd0\xb1\xd1\x80\xd1\x8c 2023       \n'
Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
UnicodeDecodeError: 'iso2022_jp' codec can't decode byte 0xd0 in position 4: illegal multibyte sequence # Вот, кстати, возможная проблема кодировок №2: не все последовательности байт вообще воспроизводимы хоть какими-то символами
```

Здесь у нас также есть возможность прыгать указателем по файлу и читать с какого-то места, н-р
Однако можно попасть в ловушку многобайтовых кодировок:

```
>>> f = open('cal', 'rt')
>>>
>>> f.seek(10)
10
>>> f.read(5)
'брь 2'
>>> f.read(6)
'023   '
>>> f.seek(9)
9
>>> f.read(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<frozen codecs>", line 322, in decode
UnicodeDecodeError: 'utf-8' codec can't decode byte 0x8f in position 0: invalid start byte
>>> # Тк кодировка двубайтовая - сдвиг, попали на середину буквы условно
```

Поговорим про Ввод-вывод

У нас есть базовые потоки ввода-вывода, а также поток ошибок. У них есть свои буферы, которые умеют читать текстовые символы и запоминают их в виде байт

```
>>> import sys
>>> sys.stdin
<_io.TextIOWrapper name='<stdin>' mode='r' encoding='utf-8'>
>>> sys.stdout
<_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>
>>> sys.stder
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'sys' has no attribute 'stder'. Did you mean: 'stderr'?
>>> sys.stderr
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'>
>>> sys.stdin.buffer
<_io.BufferedReader name='<stdin>'>
>>> sys.stdin.buffer.readline()
1qwertyuфывапро
b'1qwertyu\xd1\x84\xd1\x8b\xd0\xb2\xd0\xb0\xd0\xbf\xd1\x80\xd0\xbe\n'
```

Далее у нас есть модуль ***io*** - он обмазывает данные файловым интерфейсом, чтобы их можно было интерпретировать, как файлы. Н-р, ***StringIO*** делает как бы файлом обычные строки

```
>>> txt = "ASHGASGAKJSGLJASHDKL"
>>> from io import StringIO
>>> with StringIO(txt) as f:
...     print(f.read())
...
ASHGASGAKJSGLJASHDKL
>>> txt = "ASHGASGAKJSG\nLJASHDKL"
>>> with StringIO(txt) as f:
...     print(f.read())
...
ASHGASGAKJSG
LJASHDKL
```

Теперь такой запрос: Хотим записать float в файл

```
>>> f =1.2345
>>> ff = open('ff', 'wt')
>>> print(f, file = ff)
>>> ff.close()

stephen@The-Night-Road:~$ cat ff
1.2345
stephen@The-Night-Road:~$ hexdump -c ff
0000000   1   .   2   3   4   5  \n
0000007
```

Получилось, что мы записали строку, а не чиселко
А хочется, чтобы в файле хранились объекты а не строки

Специально для ленивых и гениальных питонистов: Сериализация - возможность в файл фигачить прямо объектами. Ну а что? Это же та же память, в которую мы такие объекты сохранять умеем. так почему бы и нет?)

Спешл вор ас нам придумали Огуречный модуль ***pickle*** для закусывания самых гениальных идей питонистов, которые без алкоголя в голову бы не пришли)))

```
>>> import pickle
>>> with open("dump", 'wb')as dump:
...     pickle.dump(1.2345, dump)
...     pickle.dump('QWEDFGHJNM', dump)
...     pickle.dump({'111': 123, "!@#": "!@#"}, dump)
...
```

Вот как выглядит этот файлик:

```
stephen@The-Night-Road:~$ hexdump -C dump
00000000  80 04 95 0a 00 00 00 00  00 00 00 47 3f f3 c0 83  |...........G?...|
00000010  12 6e 97 8d 2e 80 04 95  0e 00 00 00 00 00 00 00  |.n..............|
00000020  8c 0a 51 57 45 44 46 47  48 4a 4e 4d 94 2e 80 04  |..QWEDFGHJNM....|
00000030  95 15 00 00 00 00 00 00  00 7d 94 28 8c 03 31 31  |.........}.(..11|
00000040  31 94 4b 7b 8c 03 21 40  23 94 68 02 75 2e        |1.K{..!@#.h.u.|
0000004e
stephen@The-Night-Road:~$ hexdump -c dump
0000000 200 004 225  \n  \0  \0  \0  \0  \0  \0  \0   G   ? 363 300 203
0000010 022   n 227 215   . 200 004 225 016  \0  \0  \0  \0  \0  \0  \0
0000020 214  \n   Q   W   E   D   F   G   H   J   N   M 224   . 200 004
0000030 225 025  \0  \0  \0  \0  \0  \0  \0   } 224   ( 214 003   1   1
0000040   1 224   K   { 214 003   !   @   # 224   h 002   u   .
000004e
```

```
stephen@The-Night-Road:~$ python3.11
Python 3.11.0rc1 (main, Aug 12 2022, 10:02:14) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pickle
>>> f = open('dump', 'rb')
>>> pickle.load(f)
1.2345
>>> pickle.load(f)
'QWEDFGHJNM'
>>> pickle.load(f)
{'111': 123, '!@#': '!@#'}
>>> pickle.load(f)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
EOFError: Ran out of input
```

Хотим теперь запихивать СИшные байты - int в 4 байта, char в один итд - всё для нас, модуль ***struct***

```
>>> import struct
>>> struct.pack("bhl", 1, 2, 3) # первый параметр - форматная строка с типами вводимых далее данных
b'\x01\x00\x02\x00\x00\x00\x00\x00\x03\x00\x00\x00\x00\x00\x00\x00'
>>> len(struct.pack("bhl", 1, 2, 3)) # всё по правилам, little-endian и со сдвигами для выравнивания
16
>>> struct.pack("bhl", 1, 0x1214, 0xdeadbeef)
b'\x01\x00\x14\x12\x00\x00\x00\x00\xef\xbe\xad\xde\x00\x00\x00\x00'
>>>
>>> struct.pack("=bhl", 1, 0x1214, 0xdeadbee)
b'\x01\x14\x12\xee\xdb\xea\r'
```

Питон может и в работу с БД))) Приветствуем модуль ***dbm***

```
>>> import dbm
>>> base = dbm.open('0.db', 'c')
>>> base["QWE"] = '1'
>>> base['RTY'] = '100500'
>>> base.close()
>>> base = dbm.open('0.db', 'r')
>>> base["QWE"]
b'1'
>>> base['RTY']
b'100500'
>>> base[b'RTY']
b'100500'
```