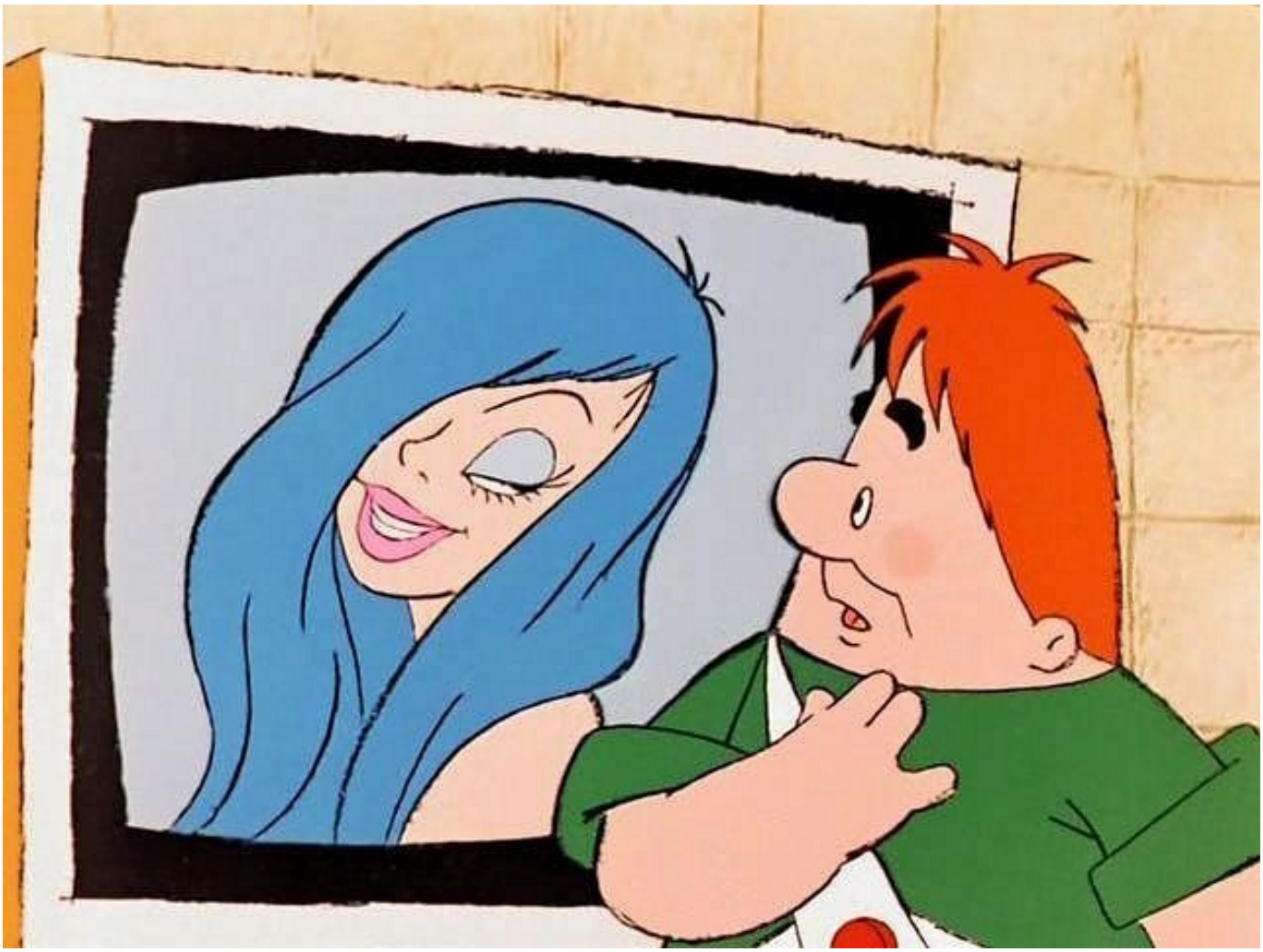
ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ НА ЯЗЫКЕ PYTHON

(Python Beginning)

«И весь Интернет – это всего лишь гигантское хранилище данных.
Без методов и инструментов контекстного поиска он никому не нужен!»



Урок 8. (Lesson 8) Основы программирования

ФАЙЛЫ

«Огромная ошибка – делать выводы, не имея необходимой информации.»
(Артур Конан Дойл»)

Несколько десятков лет назад *Янош Лайош Нейман*, венгро-американский математик, более известный миру под именем Иоган фон Нейман (***John von Neumann***) предложил архитектуру ЭВМ, получившую его имя – «Архитектура Фон-Неймана». Тогда, в далеких 1940-х года, была предложена структура ЭВМ с

общей шиной. Были определены три основных узла в составе ЭВМ: «***ЦПУ***», «***память***» и «***устройства ввода-вывода***», - из которых предлагалось строить ЭВМ. Им же, Фон-Нейманом, были сформулированы принципы, получившие название «Принципы Фон-Неймана»:

1. Принцип однородности памяти.
2. Принцип адресности.
3. Принцип программного управления.
4. Принцип двоичного кодирования.

Время шло, вычислительная техника значительно развилась и изменилась. Архитектура Фон-Неймана была развита и претерпела значительное усложнение. А сейчас какой категории «устройства ввода-вывода» или «память» можно отнести «жесткий диск»?

- На жестком диске можно хранить информацию.
- На жесткий диск можно выводить и сохранять результаты работы программ.
- Операционная система организует процесс отражения адресного пространства на жесткий диск (свопинг).

На низком, системном уровне нас интересует следующая информация:

- как организована информация на жестком диске;
- какая именно информация хранится на жестком диске;
- что такое файл и как он организован.

При изучении организации ЭВМ вы уже узнали, что носителями информации на жестком диске (HDD) являются магнитные домены. Поверхность диска разбита на дорожки, сектора. Единица измерения двоичной информации – один бит. Байт состоит их восьми бит. Минимальная единица для адресации – один байт. Логическая единица хранения информации на жестком диске – один файл. Физическая единица для построения файла на жестком диске – кластер (группа секторов). Размер кластера фиксирован и определяется способом форматирования. Файл может занимать несколько кластеров. Каждый файл имеет имя, физический размер (количество байт) и логический размер (количество кластеров). Файл может располагаться физически на нескольких жестких дисках и даже на нескольких компьютерах, объединенных в одной сетевой файловой системой. Файлы бывают «небольшие» и «очень большие», до нескольких пета байт.

С точки зрения высокоуровневого программирования нам известно, что у файла есть имя. Имя файла может включать путь к файлу. Различают абсолютный и относительный путь к файлу.

При работе с файлами в программе создается объект типа файл. Файлы-объекты – это основной интерфейс между программным кодом на языке Python и внешними файлами на компьютере. Основные методы, предназначенные для работы с файлами: открытие (`Open`), закрытие (`Close`), чтение (`Read`), запись

(*Write*), добавление (*Append*), исполнение (*eXecute*). Файл может быть открыт в текстовом режиме, либо в двоичном режиме.

Создание объекта типа файл

Для того, чтобы создать объект типа файл, необходимо вызвать метод open(), передав ему в качестве параметров строки, содержание имя файла и режим доступа к файлу. Формат операции *open()*:

```
<Ссылка_на_файловый_объект> = open(<Имя_файла>, <Режим_доступа>)
```

Например:

```
f = open("result", "w")
```

В случае успешного выполнения кода ссылка будет связана *f* с файловым объектом. В текущей директории будет открыт файл с именем "result", файл будет открыт в режиме записи (***w****rite*).

Рассмотрим следующий пример.

```
f = open("myfile", "w")        # открыть файл для вывода (создать)
f.write('Hello text file\n')   # записать в файл строку текста
f.close()                      # закрыть файл
```

В текущей для *Python*-интерпретатора будет открыт файл под именем "myfile", если файл не существует, то он будет создан (конечно, если у Пользователя, от имени которого вы запустили программу, есть на это права). Файл будет открыт текстовом режиме. В файл будет выведена/записана строка текста. После этого файл будет закрыт.

```
f = open("myfile", "r")  # открыть файл в режиме чтения
print(f.readline())  # прочитать из файла строку текста и вывести её
print(f.readline())  # прочитать из файла пустую строку: end of file
f.close()            # закрыть файл
```

Вышеприведенный пример выполняет чтение строки из ранее созданного файла. В файл были записаны две строки. Вторая строка содержит символ конца файла – ***EOF***, с которым вы уже хорошо знакомы по языку *C*.

Процедура построчного чтения файла может быть выполнена в цикле.

```
f = open("myfile", "r")  # открыть файл в режиме чтения
while 1:
    line = f.readline())  # прочитать из файла строку текста
    if not line: break    # если прочитанная строка пуста - прервать
    print(line)  # прочитанную строку можно вывести на экран ...
f.close()            # необходимо закрыть файл
```

Как вы уже догадались, в случае, если при обращении к файлу произойдет ошибка, обусловленная отсутствием файл, либо отсутствием необходимых прав доступа к файлу, в программе будет "***выброшен объект***» исключения.

Рассмотрим несколько вариантов обработки исключений.

```
import sys

def open_file(file_name, mode):
    """Open a file."""
    try:
        the_file = open(file_name, mode)
    except(IOError), e:
        print("Unable to open the file", file_name,
              "Ending program.\n", e)
        input("\n\nPress the enter key to exit.")
        sys.exit()
    else:
        return the_file

def next_line(the_file):
    """Return next line from the trivia file, formatted."""
    line = the_file.readline()
    line = line.replace("/", "\n")
    return line

trivia_file = open_file("trivia.txt", "r")
title = next_line(trivia_file)

print(title)
```

И следующий пример.

```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except IOError, (errno, strerror):
    print("I/O error(%s): %s" % (errno, strerror))
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error:", sys.exc_info()[0])
    raise
```

И Python не был бы Python, если в нем не было что-нибудь, позволяющее делать все это более изящно ☺. Для этого выполнения этой работы используется менеджер контекста *with … as*. Вот пример его использования:

```
with open('newfile.txt', 'w', encoding='utf-8') as f:
    d = int(input())
    print('1 / {} = {}'.format(d, 1 / d), file=f)
```

В задачи менеджера контекста входит безопасное открытие файла, безопасная обработка возникающих исключений и гарантированное закрытие файла при выходе интерпретатора за пределы «менеджера контекста».

В упрощенной форме «***менеджер контекста***» можно рассматривать как оператор управления. Обобщенная форма его использования:

```
with <Объект> as <Ссылка-псевдоним>:
    <Область_безопасного_использования_объекта_по_ссылке-псевдониму>
[А на данную область действия менеджера уже не распространяется]
```

В данном случае под *<Объектом>* подразумевается создаваемый объект, который может вызвать исключение. Для доступа к объекту в менеджере контекста используется *<Ссылка-псевдоним>*. Ещё один пример.

```
>>> with open('workfile') as f:
...     read_data = f.read()
>>> f.closed
True
```

Атрибуту файлового объекта *closed* присвоено значение *True*, это означает, что файл успешно закрыт.

Вернемся к режимам открытия файла (режимам доступа). Режимы доступа к файлу в *Python* совпадают с режимами доступа к файлу, определенными в языке *ANSI C*.

Таблица 1. Режимы доступа к файлу

Символ режима	Описание режима
r	Открыть для чтения (***Read***). По умолчанию файл открывается в текстовом режиме – `r[t]`.
rU либо U	Открыть для чтения с поддержкой универсальным символом новой строки *NEWLINE* (PEP 278)
W	Открыть для записи (***Write***) (создать файл, если это необходимо)
A	.Открыть для добавления (***Append***) (добавляется с позиции символа конца файла *EOF*; создать файл, если это необходимо)
r+	Открыть файл для чтения и записи
w+	Открыть файл для чтения и записи
a+	Открыть файл для чтения и записи
Rb	Открыть файл для чтения в бинарном режиме
Wb	Открыть файл для записи в бинарном режиме
ab	Открыть файл для добавления в бинарном режиме
rb+	Открыть файл для чтения и записи в бинарном режиме
wb+	Открыть файл для чтения и записи в бинарном режиме
ab+	Открыть файл для чтения и записи в бинарном режиме

Открытие файла

Подробно рассмотрим метод `open()`. Он имеет следующий формат:

```
open(<Имя_файла> [, mode='r'][, buffering=-1][, encoding=None]
     [, errors=None] [, newline=None] [, closefd=True])
```

Первый параметр метода *open()* – *<Имя_файла>* задается в виде строки. Необходимо учитывать саму структуру и состав строки, содержащей как имя файла, так и путь к нему. Это связано с тем, что в составе пути к файлу может использоваться управляющий слэш символ. В *UNIX/Linux* используется прямой слэш, а в *MS Windows* – обратный слэш. Обратный слеш в составе строки определяет управляющие *Escape*-последовательности. Поэтому символ обратного слеша в *Windows*-именах необходимо дублировать. Рассмотрим примеры правильно составленных имен файлов.

```
"C:\\temp\\myDB\\data01.txt"
r"C:\temp\myDB\data01.txt"
"C:/temp/myDB/data01.txt"
"./data02.txt"
"../data03.dat"
```

Следующий вариант имени файла будет неверным.

```
"C:\temp\myDB\data01.txt"
```

Учтите, что управляющие *Escape*-последовательности, такие, как *'\t', '\n', '\f'* могут существенно исказить имя файла, если будут случайно использованы вами при формировании имени файла.

Для ускорения работы с памятью используются механизмы буферизации. Необязательный параметр *buffering* позволяет задать размер буфера. Если в качестве данного параметра указано значение 0, то данные сразу будут записываться в файл. Данное значение допустимо в бинарном режиме доступа к файлу. Значение 1 используется при построчной записи данных в файл, оно используется только в текстовом режиме. Любое положительное значение в качестве данного параметра позволяет задать размер буфера. Отрицательное значение, используемое по умолчанию, означает установку размера буфера, принятого в данной системе по умолчанию.

При работе файлом в текстовом режиме можно указать значение используемой кодировки. По умолчанию используется кодировка *Unicode*. Например.

```
fp = open(r'source.dat', 'w', encoding='utf-8')
fp.write('Строка текста')
fp.close()
```

Основные файловые операции

Прежде чем говорить о файловых операциях замечу, что Python 3.x поддерживает следующие форматы файлов:

- *txt* – обычный текстовый файл, используемый для хранения данных ы виде набора символов, исключающий использование структурированных метаданных;
- *csv* – текстовые файлы, содержащеи «наборы» записей. Записи представляют собой последовательность строк. Каждая строка состоит ни забора полей. Поля разделены символами разделителями. Данный формат используют многие современные табличные процессоры;
- *HTML* – аайл *HyperText Markup Language* содержит структурированные данные. Данный формат используется для форматирования веб-файлов. Хотя никто не мешает вам разработать собственный набор тегов и, соответственно, расширить существующий стандарт HTML-кодирования.
- *JSON* – файл в формате *JavaScript Object Notation*. Один из наиболее часто используемых форматов структурирования метаданных. Широко используется при организации данных в сетевых хранилищах.

«Знал бы прикуп, жил бы в Сочи.»
(«народный» фольклор)

Рассмотрим операции, доступные для выполнения действий над файловыми объектами.

Таблица 2. Файловые операции

Операция	**Описание операции и формат использования**

`open(<Имя>,<Режим>)`	Открытие файла
	`out_f = open(r'C:\result', 'w')`
	`in_f = open('data','r')`
	`in_f = open('data')`

`read()`	Чтение файла целиком в виде одной строки `aStr = in_f.read()`
`read(N)`	Прочитать из файла следующие ***N*** символов. Символы следуют за текущей позицией «***файлового указателя***». `aStr = in_f.read(N)`
`readline()`	Прочитать из файла строку текста, включая символ конца строки. `aString = in_f.readline()`
`readlines()`	Прочитать файл целиком в виде списка строк. Каждая строка будет содержать в себе символ конца строки. `aStrList = in_f.readline()`
`write(aString)`	Записать в файл строку текста (или набор байт). `out_f.write(aString)`
`writelines(aStrList)`	Записать в файл указанный список строк. `out_f.writelines(aStringList)`
`close()`	Закрыть файл. `in_f.close()`
`flush()`	«Вытолкнуть» выходные файловые буферы на диск. Используется при буферизированной работе с файловым объектом. `out_f.flush()`

`seek(N)`	Изменить текущую позицию в файле (переместить позицию «файлового указателя»), смещая её на N байтов от начала файла. `out_f.seek(N)`
`tell()`	Возвращает текущее положение в файле (позицию «файлового указателя») в байтах от начала файла. `len = in_f.tell()`

.

Рассмотрим следующий пример работы с файлами – простой текстовый редактор, с минимальным набором поддерживаемых функций.

```
# -*- coding: cp1251 -*-
# filename: file_test_01.pyw
from tkinter import *
from tkinter.filedialog import *
from tkinter.messagebox import *
import fileinput

def _open():
    op = askopenfilename()
    for line in fileinput.input(op):
        txt.insert(END, line)
def _save():
    sa = asksaveasfilename()
    letter = txt.get(1.0, END)
    f = open(sa, "w")
    f.write(latter)
    f.close()
def close_window():
    if askyesno("Exit", "Do you want to quit"):
        root.destroy()
def about():
    showinfo("MiniEditor", "This is text editor.\n(mini version)")

root = Tk()
my_menu = Menu(root)
root.config(menu=my_menu)

file_menu = Menu(my_menu)
quit_menu = Menu(my_menu)
help_menu = Menu(my_menu)

my_menu.add_cascade(label="File", menu=file_menu)
file_menu.add_command(label="Open file", command=_open)
file_menu.add_command(label="Save as...", command=_save)
```

```
my_menu.add_cascade(label="Quit", menu=quit_menu)
quit_menu.add_command(label="Exit", command=close_window)

my_menu.add_cascade(label="Help", menu=help_menu)
help_menu.add_command(label="About", command=about)

txt = Text(root, width=80, height=25, font='12')
scr = Scrollbar(root, command=txt.yview)
txt.pack(side='left',expand=YES, fill=BOTH)
scr.pack(side='right', expand=YES, fill=BOTH)

root.mainloop()
```

Рассмотрим результат работы данного кода.

Литература и источники в Интрнет

1. Мэтиз Эрик. Изучаем Python. Программирование игр, визуализация данных, веб-приложения. [Текст] – СПб.: Питер, 2017. – 496 с.
2. Любавич Билл. Простой Python. Современный стиль программирования. [Текст] – СПб.: Питер, 2016. – 480 с.
3. Лутц М. Изучаем Pyhon, 4-е издание. – Пер. с англ. [Текст] – СПб.: СимволПлюс, 2011. – 1280 с.
4. Прохоренок Н.А. Python 3 и PyQt 5. Разработка приложений [Текст] / Н.А. Прохоренок, В.А. Дронов. – СПб.: БХВ-Петербург, 2016. – 832.: ил.