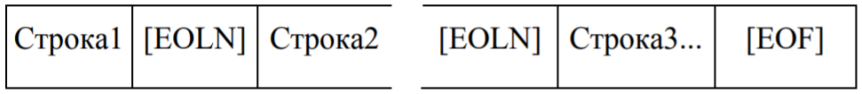
Лабораторная работа №7

Работа с текстовыми файлами. Строки.

Python позволяет работать с текстовыми файлами. **Текстовые файлы** содержат текстовую информацию в виде отдельных порций, называемых **строками**. Строки состоят из отдельных символов и могут иметь различную длину.

Для отделения строк друг от друга используется специальный символ «конец строки» (End Of LiNe, EOLN), который записывается в файл автоматически при записи символа **\n** (в случае программного заполнения файла). В конце файла записывается символ «конец файла» (End Of File, EOF). Учитывая, что информация на диске располагается последовательно – байт за байтом, общую структуру текстовых файлов можно изобразить следующим образом:



Учитывая, что «строка» – понятие неоднозначное (в ней может быть как один символ, так и несколько тысяч), работа с текстовыми файлами осуществляется методом последовательного доступа: чтобы добраться до нужной порции информации (строки), необходимо прочитать все предыдущие строки от начала файла до нужного места.

Python поддерживает множество различных типов файлов, но условно их можно разделить на два виде: текстовые и бинарные.

Текстовые файлы – это, к примеру, файлы с расширением **cvs, txt, html**, в общем любые файлы, которые сохраняют информацию в текстовом виде.

Бинарные файлы - это изображения, аудио и видеофайлы и т.д.

В зависимости от типа файла работа с ним может немного отличаться.

Технология работы с текстовыми файлами.

При работе с текстовыми файлами необходимо выполнять следующую последовательность действий.

Первый шаг – это открытие файла с помощью функции **open**. Функция **open** возвращает ссылку на файловый объект, которую нужно записать в переменную, чтобы потом через данный объект использовать методы ввода-вывода:

f = open(<file>, <mode>)

<file> - путь к файлу. Путь файла может быть **абсолютным**, то есть начинаться с буквы диска, например, *C://Temp/namefile.txt*. Либо можно быть **относительным**, например, *Temp/namefile.txt* - в этом случае поиск файла будет идти относительно расположения запущенного скрипта Python.

<mode> - устанавливает режим открытия файла в зависимости от того, что необходимо сделать с ним.

Возможны следующие режимы:

Режим	Описание
'r'	открытие на чтение (является значением по умолчанию); если файл не найден, то генерируется исключение FileNotFoundError
'w'	открытие на запись; если файл отсутствует, то он создается; если подобный файл уже есть, то он создается заново, и соответственно старые данные в нем стираются.
'x'	открытие на запись, если файла не существует, иначе исключение.
'a'	открытие на дозапись, информация добавляется в конец файла; если файл отсутствует, то он создается.
'b'	открытие в двоичном режиме.
't'	открытие в текстовом режиме (является значением по умолчанию).
'+'	открытие на чтение и запись.

Режимы могут быть объединены, то есть, к примеру, **'rb'** - чтение в двоичном режиме. По умолчанию режим равен **'rt'**.

Пример. Открытие файла на для записи:

```
f = open('hello.txt', 'w')
```

Второй шаг – это собственно работа с файлом.

«Работа» – это прежде всего чтение символов/строк из файла и запись символов/строк в файл. Чтение из текстового файла может быть реализовано одним из следующих способов.

Чтение файла

1. Метод **read()** считывает все содержимое из файла и возвращает строку, которая может содержать символы '\n'. Если методу **read** передать целочисленный параметр (read(n)), то будет считано не более (n) заданного количества символов.

```
# метод read без параметров считывает ВЕСЬ файл

f = open('hello.txt','r')
line = f.read()
print(line, end='')
f.close() # закрыть файл
```

```
# метод read с параметром считывается 10 первых символов
f = open('hello.txt','r')
line = f.read(10)
print(line, end='')
f.close() # закрыть файл
```

2. Метод **readline()** считывает одну строку из файла (до символа конца строки ‘\n’, возвращается считанная строка вместе с символом ‘\n’, поэтому для удаления символа ‘\n’ из конца файла удобно использовать метод строки **rstrip()**). Если считывание не было успешно (достигнут конец файла), то возвращается пустая строка.

```
# метод readline

f = open('hello.txt','r')
line = f.readline()
print(line, end='')
f.close() # закрыть файл
```

3. Метод **readlines()** считывает все строки из файла и возвращает список из всех считанных строк (одна строка — один элемент списка). При этом символы ‘\n’ остаются в концах строк.

При чтении файла можно столкнуться с тем, что его кодировка не совпадает с ASCII. В этом случае мы явным образом можем указать кодировку с помощью параметра **encoding**:

```
filename = ‘hello.txt’
with open(filename, encoding=’utf8’) as file:
    text = file.read()
```

Конструкция **with**

При открытии файла или в процессе работы с ним мы можем столкнуться с различными исключениями, например, к нему нет доступа и т.д. В этом случае программа выдаст в ошибку, а ее выполнение не дойдет до вызова метода close, и соответственно файл не будет закрыт.

В этом случае мы можем обрабатывать исключения

```
try:
    f = open(‘hello.txt’, ‘w’)
    try:
        f.write(‘hello world’)
    except Exception as e:
        print(e)
    finally:
        f.close()
except Exception as ex:
```

```
    print(ex)
```

В данном случае вся работа с файлом идет во вложенном блоке **try**. И если вдруг возникнет какое-либо исключение, то в любом случае в блоке **finally** файл будет закрыт.

Однако есть и более удобная конструкция - конструкция **with**:

```
with open(<file>, <mode>) as file_obj:
    инструкции
```

Эта конструкция определяет для открытого файла переменную file_obj и выполняет набор инструкций. После их выполнения файл автоматически закрывается. Даже если при выполнении инструкций в блоке **with** возникнут какие-либо исключения, то файл все равно закрывается.

Так предыдущий пример выглядеть следующим образом:

```
with open('hello.txt', 'w') as f:
    f.write('hello world')
```

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. Запишем некоторую информацию в файл 'hello.txt':

```
with open('hello.txt', 'w') as f:
    f.write('hello world')
```

Теперь дозапишем в этот файл еще одну строку:

```
with open('hello.txt', 'w') as f:
    f.write('\ngood bye, world')
```

Работа с файлами **csv.**

Одним из распространенных типов файлов, которые хранят в удобном виде информацию, является формат **csv**. Каждая строка в файле **csv** представляет отдельную запись или строку, которая состоит из отдельных столбцов, разделенных запятыми. Но хотя формат csv - это формат текстовых файлов, Python для упрощения работы с ним предоставляет специальный встроенный модуль **csv**. Он содержит свой набор методов.

Третий шаг – закрытие файла методом **close**.

```
# метод close

f = open('hello.txt','w') # открыпm файл на запись

f.close() # закрыть файл
```

Обратите внимание, что объекту **f** (имя может быть любым) на самом деле определяет Буфер в оперативной памяти, в который грузится информация из файла. Вся работа с данными реализуется непосредственно в этом Буфере.

При записи информации в файл реально идет занесение данных в Буфер в оперативной памяти. Метод f.close() сбрасывает содержимое Буфера в файл на диске, записывает символ EOF и освобождает Буфер.

При чтении также идет работа с Буфером. Метод f.close() в этом случае просто освобождает оперативную память. Если файл не закрыть, Буфер останется в памяти, что нежелательно.

Методы строк.

При работе с файлами можно использовать методы строк для эффективной работы.

Строка состоит из последовательности символов. Тип строки **str**. В языке Python нет отдельного символьного типа. Символ — это просто строка длины 1. Строка считывается со стандартного ввода функцией input().

Работа со строками во многом схожа с работой со списками. Индексация элементов в строке начинается с нуля, поэтому первый символ строки будет иметь индекс 0. А если будет попытка обратиться к индексу, которого нет в строке, то будет получено исключение IndexError. Чтобы получить доступ к символам, начиная с конца строки, можно использовать отрицательные индексы. Так, индекс -1 будет представлять последний символ, а -2 - предпоследний символ и так далее.

Получение подстроки.

При необходимости мы можем получить из строки не только отдельные символы, но и подстроку. Для этого используется следующий синтаксис:

string[:end]: извлекается последовательность символов начиная с 0-го индекса по индекс **end;**

string[start:end]: извлекается последовательность символов начиная с индекса **start** по индекс **end;**

string[start:end:step]: извлекается последовательность символов начиная с индекса **start** по индекс **end** через шаг **step.**

Функции ord и len

Поскольку строка содержит символы Unicode, то с помощью функции **ord()** можно получить числовое значение для символа в кодировке Unicode.

Для получения длины строки можно использовать функцию **len():**

```
string = "hello world"
length = len(string)
print(length)    # 11
```

Методы строк, которые можно применить в приложениях:

isalpha(): возвращает **True**, если строка состоит только из алфавитных символов
islower(): возвращает **True**, если строка состоит только из символов в нижнем регистре
isupper(): возвращает **True**, если все символы строки в верхнем регистре
isdigit(): возвращает **True**, если все символы строки - цифры
isnumeric(): возвращает **True**, если строка представляет собой число
startswith(str): возвращает **True**, если строка начинается с подстроки str
endswith(str): возвращает **True**, если строка заканчивается на подстроку str
lower(): переводит строку в нижний регистр
upper(): переводит строку в верхний регистр
title(): начальные символы всех слов в строке переводятся в верхний регистр
capitalize(): переводит в верхний регистр первую букву только самого первого слова строки
lstrip(): удаляет начальные пробелы из строки
rstrip(): удаляет конечные пробелы из строки
strip(): удаляет начальные и конечные пробелы из строки
ljust(width): если длина строки меньше параметра **width**, то справа от строки добавляются пробелы, чтобы дополнить значение **width**, а сама строка выравнивается по левому краю
rjust(width): если длина строки меньше параметра **width**, то слева от строки добавляются пробелы, чтобы дополнить значение **width**, а сама строка выравнивается по правому краю
center(width): если длина строки меньше параметра **width**, то слева и справа от строки равномерно добавляются пробелы, чтобы дополнить значение **width**, а сама строка выравнивается по центру
find(str[, start [, end]): возвращает индекс подстроки в строке. Если подстрока не найдена, возвращается число -1
replace(old, new[, num]): заменяет в строке одну подстроку на другую
split([delimeter[, num]]): разбивает строку на подстроки в зависимости от разделителя
join(strs): объединяет строки в одну строку, вставляя между ними определенный разделитель