# Работа с файлами в Python

Для вопросов по курсу:
natalya.shevskaya@moevm.info
Префикс в теме письма [CS_23XX]

*Шевская Наталья Владимировна*
*СПбГЭТУ “ЛЭТИ”, ФКТИ, МОЭВМ*

# Типы файлов

- текстовые файлы (.txt, .docx и др.)
- видео-файлы (.mp4, .webM и др.)
- аудио-файлы (.mp3, .flac, .wav и др.)
- изображения (.bmp, .png, .jpeg и др.)

# Открытие файлов

Встроенная функция open для открытия файлов *open*:

```
file_obj = open(<путь к файлу>)
```

Пример:

```
file_obj = open("test.txt")

print(file_obj)
```

Увидим на экране информацию о файле-объекте:

```
<_io.TextIOWrapper name='test.txt' mode='r' encoding='UTF-8'>
```

# Режимы при работе с текстовыми файлами

| | |
|---|---|
| `r` | Открыть файл на чтение (**по умолчанию**) |
| `w` | Открывает файл для записи. Создает новый файл, если он не существует, или усекает файл, если он существует. |
| `x` | Открывает файл для монопольного создания. Если файл уже открыт, операция завершится ошибкой. |
| `a` | Открывает файл для добавления в конец файла без его усечения. Создает новый файл, если он не существует. |
| `t` | Открытие файла в текстовом режиме (**по умолчанию**) |
| `b` | Открытие файла в бинарном виде |
| `+` | Открывает файл для обновления (чтение и запись) |

# Примеры

```
f1 = open("test.txt")       # эквивалентно использованию режимов 'r' или 'rt'

f2 = open("test.txt", 'w')  # открытие файла на запись в текстовом режиме

f3 = open("img.bmp", 'r+b') # открытие файла на чтение в бинарном режиме
```

# Выбор кодировки при открытии файла

Обычно в Windows используется кодировка `cp1252`

В то время как в Linux: `utf-8`

Пример:

```
f4 = open("test.txt", mode='r', encoding='utf-8')
```

# Закрытие файла

Для закрытия файлов применяется встроенная функция *close*.

Вы можете вызывать ее после того как все необходимые действия с файлом и данными из него будут завершены:

```
f = open("test.txt", encoding='utf-8')
# здесь могут быть ваши операции с файлом и данными из него
f.close()
```

**! Файл надо закрыть обязательно ! Об этом следует помнить !**

# Закрыть файл нужно при любом исходе

Используется обработчик исключительных ситуаций *try+finally*:

```
try:
  f = open("test.txt", encoding = 'utf-8')
  # если файл открылся без проблем, этот код будет выполняться
finally:
  f.close()
```

# Автоматическое закрытие файла

Конструкция *with* создает специальную область видимости, при выходе из которой файл будет закрыт (функция *close* вызывается без нашего участия):

Пример:

```
with open("test.txt", encoding = 'utf-8') as f:
  # здесь могут быть ваши операции с файлом и данными из него
```

# Запись текста в файл -- метод write

Для того чтобы записать текст в файл, используется метод объекта-файла write.

Пример записи нескольких строчек в файл:

```
with open("test.txt", 'w', encoding='utf-8') as f:
    f.write("Первая строка\n")
    f.write("Вторая строка и за ней еще одна пустая\n\n")
    f.write("Ну, и последняя строчка!\n")
```

# Метод файлов write (2)

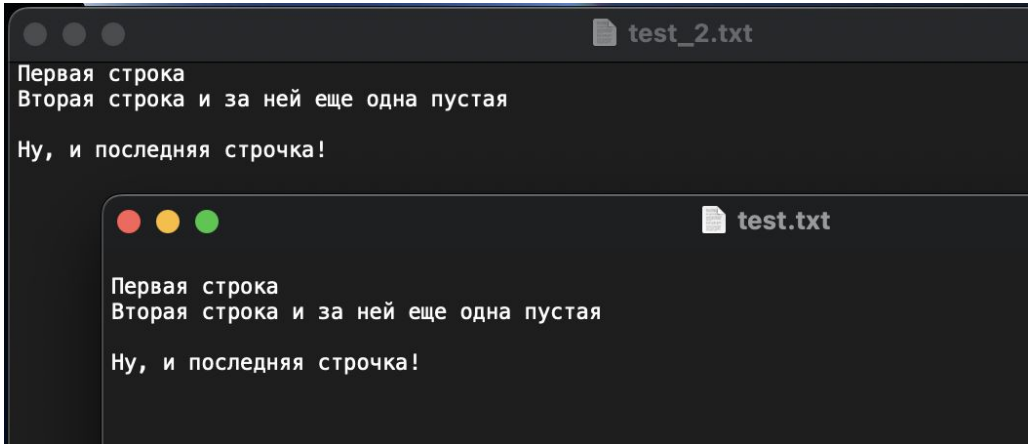
Записывать в текст можно также многострочный литерал:

```
literal = '''Первая строка
Вторая строка и за ней еще одна пустая

Ну, и последняя строчка!
'''
with open("test_2.txt", 'w', encoding='utf-8') as f:
    f.write(literal)
```

# Результат двух предыдущих примеров

Результат в обоих случаях будет одинаковым:



# Чтение данных из файла

Метод *read(size)*, где *size* -- это сколько данных следует прочитать из файла (количество символов).

Пример с файлом *test_2.txt* из предыдущего примера:

```
with open("test_2.txt", 'r', encoding='utf-8') as f:
    data = f.read(10)
    print(data, type(data), len(data), sep='\n')
```

Увидим на экране:

```
Первая стр
<class 'str'>
10
```

# Метод *read()* без аргумента *size*

По умолчанию *size* принимает значение, сколько всего символов в файле. Если не указывать *size*, будут прочитаны все символы до конца:

```
with open("test_2.txt", 'r', encoding='utf-8') as f:
    data = f.read()
    print(data, type(data), len(data), sep='\n')
```

Увидим на экране:

```
Первая строка
Вторая строка и за ней еще одна пустая

Ну, и последняя строчка!
<class 'str'>
79
```

# Метод *read(size)*

Если продолжить чтение из файла, когда данные в нем закончились, методы *read* будет возвращать пустые строки.

Упражнение: проверьте это на практике

# Прочитать одну строку из файла

- Если мы не знаем размер (количество символов) в одной строке в файле, но знаем, что файл заполнен построчно, то нам было бы удобно считывать каждую строку в отдельности.
- Например, в файле построчно записаны элементы квадратной матрицы через пробел, размер матрицы не известен.
- Для этой задачи можно использовать метод *readline*

# Метод readline

```
with open("test_2.txt", 'r', encoding='utf-8') as f:
    line = f.readline()
    print(line)
```

Вывод:

```
Первая строка
```

Если вызвать readline уже после того, как будет прочитана последняя строка файла, метод вернет пустую строку.

# Прочитать список строк из файла

- Используя метод readline можно считывать строки из файла и складывать их в список.
- Используя read без size можно прочитать весь файл в один объект-str, выполнить strip(‘\n’), и получить список строк.
- А можно использовать встроенный метод readlines, который в буквальном смысле возвращает список строк из файла

# Метод *readlines*

```
with open("test_2.txt", 'r', encoding='utf-8') as f:
    line_list = f.readlines()
    print(line_list)
```

Увидим на экране:

```
['Первая строка\n', 'Вторая строка и за ней еще одна пустая\n', '\n', 'Ну, и последняя строчка!\n']
```

Обратите внимание: символ '\n' -- это часть строки! Не забудьте от него избавиться через метод *strip*, чтобы не было неприятных сюрпризов. Возможно, после strip потребуется убрать пустые строки из этого списка.

# Еще немножно полезных методов

Если мы использовали *read(size)*, например, считывали посимвольно и в какой-то момент забыли, где сейчас в файле находится курсор, то можно узнать текущую позицию курсора в файле с помощью метода *tell*:

```
with open("test_2.txt", 'r', encoding='utf-8') as f:
    data = f.read(10)
    print(data, f.tell(), sep='\n')
```

Увидим на экране:

```
Первая стр
19
```

Не пугайтесь! Метод tell возвращает текущую позицию **в байтах**.

# Еще немножно полезных методов (2)

Если нам известно, что на какой-то определенной позиции что-то находится, и нам нужно от этой позиции прочитать какое-то количество символов, то чтобы переместить курсор в нужную позицию в текста файла есть метод *seek*:

```
with open("test_2.txt", 'r', encoding='utf-8') as f:
    f.seek(13)  # позицию надо указывать в байтах
    data = f.read(6)
    print(data, sep='\n')
```

Увидим:

```
строка
```

# Полезные ссылки

- Python File I/O -- (https://www.programiz.com/python-programming/file-operating)