Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

Лабораторная работа № 4

“Исследование протоколов, форматов обмена информацией и языков разметки документов”

Вариант № 14

Выполнил:

Сандов Кирилл Алекссевич

Группа:

P3113

Проверил:

к.т.н преподаватель Белозубов Александр Владимирович

г. Санкт-Петербург

2022

Оглавление

Задание

Обязательное задание:

Составить файл с расписанием занятий своей учебной группы в указанный день в исходном формате, написать программу на языке Python 3.x, которая бы осуществляла парсинг и конвертацию исходного файла в новый.

День недели: среда

Исходный формат: JSON

Результирующий формат: YAML

Нельзя использовать готовые библиотеки, в том числе регулярные выражения в Python и библиотеки для загрузки XML-файлов.

Дополнительное задание № 1:

1. Найти готовые библиотеки, осуществляющие аналогичный парсинг и конвертацию файлов.
2. Переписать исходный код, применив найденные библиотеки. Регулярные выражения также нельзя использовать.
3. Сравнить полученные результаты и объяснить их сходство/различие.

Дополнительное задание № 2:

1. Переписать исходный код, добавив в него использование регулярных выражений.
2. Сравнить полученные результаты и объяснить их сходство/различие.

Дополнительное задание № 3:

1. Используя свою исходную программу из обязательного задания, программу из дополнительного задания № 1 и программу из дополнительного задания № 2, сравнить стократное время выполнения парсинга + конвертации в цикле.
2. Проанализировать полученные результаты и объяснить их сходство/различие.

Дополнительное задание № 4:

1. Переписать исходную программу, чтобы она осуществляла парсинг и конвертацию исходного файла в любой другой формат (кроме JSON, YAML, XML, HTML): PROTOBUF, TSV, CSV, WML и т.п.
2. Проанализировать полученные результаты, объяснить особенности использования формата.

Обязательное задание

Сначала был создан JSON-файл, содержащий расписание предметов на среду:

schedule.json

```
{
    "day": "Среда",
    "subjects": [
        {
            "subjectName": "Информатика (лаб)",
            "beginTime": "11:40",
            "endTime": "13:10",
            "weekNumbers": [2, 4, 6, 8, 10, 12, 14, 16],
            "auditorium": "2308 (бывш. 306) ауд.",
            "place": "Кронверкский пр., д.49, лит.А",
            "isEvenWeek": 1,
            "teacher": "Белозубов Александр Владимирович",
            "format": "очно-дистанционный"
        },
        {
            "subjectName": "Информатика (лаб)",
            "beginTime": "13:30",
            "endTime": "15:00",
            "weekNumbers": [2, 4, 6, 8, 10, 12, 14, 16],
            "auditorium": "2308 (бывш. 306) ауд.",
            "place": "Кронверкский пр., д.49, лит.А",
            "isEvenWeek": 1,
            "teacher": "Белозубов Александр Владимирович",
            "format": "очно-дистанционный"
        }
    ]
}
```

Конвертация данных из формата JSON в формат YAML будет выполнена с помощью следующих шагов:

1. Чтение JSON-файла и сохранение его в виде «сырого» текста
2. Парсинг этого текста и его интерпретация с помощью объектов языка Python. Причём достаточно будет следующих объектов: dict, list, int, str.
3. Сериализация полученной совокупности объектов в строку, соответствующую структуре YAML-файла.
4. Создание нового YAML-файла и запись этой строки в него.

Чтение и парсинг JSON-файла выполняет модуль *json_parser.py*:

json_parser.py

```
import string

def parse_json_file(filename, use_regexps=False):
    """
```

```
        Parses given file and returns a python object (list or dict).
        If there's any problems, it returns None.
    """
    text = read_data(filename)

    result = None
    try:
        if use_regexps:
            result = recursive_parse_with_regexps(text)
        else:
            result = recursive_parse(text)
    except ParseError:
        print('Ошибка при попытке парсинга')

    return result


def recursive_parse(text, parent=None):
    text = text.strip()
    if parent is None:
        if text[0] == '{':
            return recursive_parse(text[1:-1], {})
        elif text[0] == '[':
            return recursive_parse(text[1:-1], [])
        else:
            raise ParseError

    elif isinstance(parent, list):
        content = special_split(text, ',')

        for item in content:
            item = item.strip()
            if item[0] == '\"':
                parent.append(item[1:-1])
            elif item[0] in string.digits:
                parent.append(int(item))
            elif item[0] == '{':
                parent.append(recursive_parse(item[1:-1], {}))
            elif item[0] == '[':
                parent.append(recursive_parse(item[1:-1], []))
            else:
                raise ParseError

        return parent

    elif isinstance(parent, dict):
        pairs = special_split(text, ',')

        for pair in pairs:
            key, value = special_split(pair, ':')
            key = key.strip()[1:-1]
            value = value.strip()

            if value[0] == '\"':
                parent[key] = value[1:-1]
            elif value[0] in string.digits:
                parent[key] = int(value)
            elif value in ['false', 'true']:
                parent[key] = bool(value)
            elif value == 'null':
                parent[key] = None
            elif value[0] == '{':
```

```
                parent[key] = recursive_parse(value[1:-1], {})
            elif value[0] == '[':
                parent[key] = recursive_parse(value[1:-1], [])
            else:
                raise ParseError

        return parent

    else:
        raise ParseError


def recursive_parse_with_regexps(text, parent=None):
    # Эта функция будет реализована в доп. задании 2


def find_closing_index(text, shift=0):
    bracket = text[0]
    if bracket not in '[{':
        raise ParseError
    closing = ']' if bracket == '[' else '}'
    level = 1
    for i in range(1, len(text)):
        c = text[i]
        if c in '[{':
            level += 1
        elif c in ']}':
            level -= 1
            if level == 0:
                if closing != c:
                    raise ParseError
                return i + shift

    return -1


def special_split(text, divider):
    result = []
    start = 0
    i = 0
    last_divider = -1
    while i < len(text):
        if text[i] == '\"':
            end = text.index('\"', i + 1)
            i = end
        elif text[i] in '{[':
            end = find_closing_index(text[i:], i)
            i = end
        elif text[i] == divider:
            result.append(text[start:i])
            last_divider = i
            start = i + 1
        i += 1

    if last_divider != -1:
        result.append(text[last_divider + 1:])
    else:
        result.append(text)

    return result
```

```
def read_data(filename):
    try:
        f = open(filename, 'r', encoding='utf-8')
    except FileNotFoundError:
        print(f'Файла \"{filename}\" не существует')
        return None

    text = f.read().strip()
    f.close()
    return text


class ParseError(Exception):
    def __init__(self, *args: object) -> None:
        super().__init__(*args)
```

Сериализацию данных в формат YAML выполняет модуль *yaml_serializer.py*:

yaml_serializer.py

```
def serialize_data(data, filename='data.yml'):
    result = None
    try:
        result = recursive_serialize(data)
    except SerializeError:
        print('Ошибка при сериализации')
        return False

    try:
        write_data(result, filename)
    except Exception:
        print('Ошибка при записи данных')
        return False

    return True


def recursive_serialize(node, tabs=0, is_in_list=False):
    if isinstance(node, bool):
        return f'{node}'.lower()

    elif isinstance(node, int) or isinstance(node, float):
        return f'{node}'

    elif node is None:
        return 'null'

    elif isinstance(node, str):
        return f'\"{node}\"'

    elif isinstance(node, dict):
        result = ''
        is_first = True
        for key, value in node.items():
            if isinstance(value, dict) or (isinstance(value, list) and not
is_short_list(value)):
                if is_first and is_in_list:
                    result += f'{key}:\n{recursive_serialize(value, tabs + 1)}\n'
```

```
                    is_first = False
                else:
                    result += f'{tab()*tabs}{key}:\n{recursive_serialize(value,
tabs + 1)}\n'
            else:
                if is_first and is_in_list:
                    result += f'{key}: {recursive_serialize(value, tabs + 1)}\n'
                    is_first = False
                else:
                    result += f'{tab()*tabs}{key}: {recursive_serialize(value, tabs
+ 1)}\n'

        if result[-1] == '\n':
            result = result[:-1]

        return result

    elif isinstance(node, list):
        result = ''
        is_first = True
        is_short = is_short_list(node)
        if is_short:
            result = '['

        for index, item in enumerate(node):
            if is_short:
                result += f'{recursive_serialize(item, tabs + 1, True)}'
                result += ', ' if index != len(node) - 1 else ']'
            else:
                if is_first and is_in_list:
                    result += f'- {recursive_serialize(item, tabs + 1, True)}\n'
                    is_first = False
                else:
                    result += f'{tab()*tabs}- {recursive_serialize(item, tabs + 1,
True)}\n'

        if result[-1] == '\n':
            result = result[:-1]

        return result

    else:
        raise SerializeError


def is_short_list(lst):
    is_short = True
    for item in lst:
        if not (
            isinstance(item, int) or \
            (isinstance(item, str) and len(item) <= 50)
        ):
            is_short = False
            break

    return is_short


def tab():
    return '  '
```

```
def write_data(data, filename):
    with open(filename, mode='w', encoding='utf-8') as f:
        f.write(data)


class SerializeError(Exception):
    def __init__(self, *args: object) -> None:
        super().__init__(*args)
```

Запуск всей необходимой последовательности функций происходит в модуле *main.py:*

main.py

```
from json_parser import parse_json_file
from yaml_serializer import serialize_data

def main_task():
    """Конвертировать schedule.json в schedule.yml"""
    data = parse_json_file('schedule.json')
    serialize_data(data, filename='schedule.yml')


if __name__ == '__main__':
    main_task()
```

После запуска программа создаст файл *schedule.yml* в своей директории. Его содержимое:

schedule.yml

```
day: "Среда"
subjects:
  - subjectName: "Информатика (лаб)"
    beginTime: "11:40"
    endTime: "13:10"
    weekNumbers: [2, 4, 6, 8, 10, 12, 14, 16]
    auditorium: "2308 (бывш. 306) ауд."
    place: "Кронверкский пр., д.49, лит.А"
    isEvenWeek: true
    teacher: "Белозубов Александр Владимирович"
    format: "очно-дистанционный"
  - subjectName: "Информатика (лаб)"
    beginTime: "13:30"
    endTime: "15:00"
    weekNumbers: [2, 4, 6, 8, 10, 12, 14, 16]
    auditorium: "2308 (бывш. 306) ауд."
    place: "Кронверкский пр., д.49, лит.А"
    isEvenWeek: true
    teacher: "Белозубов Александр Владимирович"
    format: "очно-дистанционный"
```

Дополнительное задание № 1

Для парсинга JSON-файла подойдёт поставляемая в стандартном наборе Python 3 библиотека *json*. Для сериализации данных в YAML-файл можно использовать библиотеку PyYAML. Её требуется дополнительно установить следующим образом:

```
pip install pyyaml
```

Воспользуемся этими библиотеками и выполним конвертацию файла *schedule.json*:

main.py

```
from json_parser import read_data
from yaml_serializer import write_data
import json
import yaml

def extra_task_1():
    """Конвертировать schedule.json в schedule_with_lib.yml с использованием
библиотек"""
    text = read_data('schedule.json')
    data = json.loads(text)
    serialized = yaml.dump(data, allow_unicode=True)
    write_data(serialized, filename='schedule_with_lib.yml')

if __name__ == '__main__':
    extra_task_1()
```

Результат конвертации доступен в файле schedule_with_lib.yml:

```
day: Среда
subjects:
- auditorium: 2308 (бывш. 306) ауд.
  beginTime: '11:40'
  endTime: '13:10'
  format: очно-дистанционный
  isEvenWeek: true
  place: Кронверкский пр., д.49, лит.А
  subjectName: Информатика (лаб)
  teacher: Белозубов Александр Владимирович
  weekNumbers:
  - 2
  - 4
  - 6
  - 8
  - 10
  - 12
  - 14
  - 16
- auditorium: 2308 (бывш. 306) ауд.
  beginTime: '13:30'
  endTime: '15:00'
  format: очно-дистанционный
```

```
isEvenWeek: true
place: Кронверкский пр., д.49, лит.А
subjectName: Информатика (лаб)
teacher: Белозубов Александр Владимирович
weekNumbers:
- 2
- 4
- 6
- 8
- 10
- 12
- 14
- 16
```

Отличие этого результата от результата программы, написанной вручную, состоит в следующем:

1. Порядок вывода ключей в парах. Здесь они сортируются лексикографически, а в первой программе – выводятся в том порядке, в котором они были в исходном файле.
2. Все списки выводятся в столбец, а в первой программе была проверка, является ли список коротким, чтобы выводить его в строчном формате. Так, для поля *weekNumbers* там вывод был не в столбец, а в строку.
3. Значения строк не берутся в кавычки. Однако в YAML строки могут задаваться как в кавычках (и в двойных, и в одинарных), так и без них.

Проанализировав различия, видно, что выводы обеих программ отличаются лишь в некоторых стилистических моментах, которые не влияют на корректность их результатов.

Дополнительное задание № 2

Напишем новую функцию *recursive_parse_with_regexps()*, в которой для определения типов данных будут использоваться регульрные выражения.

json_parser.py

```
import string
import re

def parse_json_file(filename, use_regexps=False):
    """
        Parses given file and returns a python object (list or dict).
        If there's any problems, it returns None.
    """
    text = read_data(filename)

    result = None
    try:
        if use_regexps:
            result = recursive_parse_with_regexps(text)
        else:
            result = recursive_parse(text)
    except ParseError:
        print('Ошибка при попытке парсинга')

    return result


def recursive_parse(text, parent=None):
      # Её реализация приведена в обязательном задании

def recursive_parse_with_regexps(text, parent=None):
    text = text.strip()
    if parent is None:
        if re.match(r'^{', text):
            return recursive_parse_with_regexps(text[1:-1], {})
        elif re.match(r'^[', text):
            return recursive_parse_with_regexps(text[1:-1], [])
        else:
            raise ParseError

    elif isinstance(parent, list):
        content = special_split(text, ',')

        for item in content:
            item = item.strip()
            if re.match(r'^\".*\"$', item):
                parent.append(item[1:-1])
            elif re.match(r'(-?(?:0|[1-9]\d*)(?:\.\d+)?(?:[eE][+-]?\d+)?)\s*(.*)',
item):
                parent.append(int(item))
            elif re.match(r'^\{', item):
                parent.append(recursive_parse_with_regexps(item[1:-1], {}))
            elif re.match(r'^\[', item):
                parent.append(recursive_parse_with_regexps(item[1:-1], []))
            else:
```

```
                raise ParseError

        return parent

    elif isinstance(parent, dict):
        pairs = special_split(text, ',')

        for pair in pairs:
            key, value = special_split(pair, ':')
            key = key.strip()[1:-1]
            value = value.strip()

            if re.match(r'^\".*\"$', value):
                parent[key] = value[1:-1]
            elif re.match(r'(-?(?:0|[1-9]\d*)(?:\.\d+)?(?:[eE][+-]?\d+)?)\s*(.*)',
value):
                parent[key] = eval(value)
            elif re.match(r'^(true|false)$', value):
                parent[key] = eval(value.capitalize())
            elif re.match(r'^null$', value):
                parent[key] = None
            elif re.match(r'^\{', value):
                parent[key] = recursive_parse_with_regexps(value[1:-1], {})
            elif re.match(r'^\[', value):
                parent[key] = recursive_parse_with_regexps(value[1:-1], [])
            else:
                raise ParseError

        return parent

    else:
        raise ParseError


def find_closing_index(text, shift=0):
    # Её реализация приведена в обязательном задании


def special_split(text, divider):
    # Её реализация приведена в обязательном задании

def read_data(filename):
      # Её реализация приведена в обязательном задании


class ParseError(Exception):
    # Его реализация приведена в обязательном задании
```

main.py

```
from json_parser import parse_json_file
from yaml_serializer import serialize_data

def extra_task_2():
    """Конвертировать schedule.json в schedule_with_regexps.yml с использованием
regexps"""
    data = parse_json_file('schedule.json', use_regexps=True)
    serialize_data(data, filename='schedule_with_regexps.yml')

if __name__ == '__main__':
```

```
    extra_task_2()
```

Результат сохранён в файл *schedule_with_regexps.yml.*

schedule_with_regexps.yml

```yaml
day: "Среда"
subjects:
  - subjectName: "Информатика (лаб)"
    beginTime: "11:40"
    endTime: "13:10"
    weekNumbers: [2, 4, 6, 8, 10, 12, 14, 16]
    auditorium: "2308 (бывш. 306) ауд."
    place: "Кронверкский пр., д.49, лит.А"
    isEvenWeek: true
    teacher: "Белозубов Александр Владимирович"
    format: "очно-дистанционный"
  - subjectName: "Информатика (лаб)"
    beginTime: "13:30"
    endTime: "15:00"
    weekNumbers: [2, 4, 6, 8, 10, 12, 14, 16]
    auditorium: "2308 (бывш. 306) ауд."
    place: "Кронверкский пр., д.49, лит.А"
    isEvenWeek: true
    teacher: "Белозубов Александр Владимирович"
    format: "очно-дистанционный"
```

Никаких различий с файлом из обязательного задания нет.

Дополнительное задание № 3

Для замера времени многократного запуска функций конвертации будем использовать функцию *timeit* из встроенной библиотеки *timeit.*

Её синтаксис:

```
timeit: (stmt: str, setup: str, number: int) -> float
```

Здесь *stmt* – это строка кода, которую необходимо замерить по времени. В нашем случае это вызов одной из функций из *main.py;*

setup – это строка кода, которую необходимо выполнить перед замером времени. Мы будем её использовать, чтобы получить функции из *main.py*;

number – количество запусков строки. Для стократного повторения каждой функции передадим значение 100.

Функция вернёт вещественное значение – время работы цикла с повторениями в секундах.

main.py

```
import timeit

def main_task():
      # Её реализация приведена в обязательном задании

def extra_task_1():
      # Её реализация приведена в доп. задании 1

def extra_task_2():
      # Её реализация приведена в доп. задании 2

def extra_task_3():
    """Сравнить стократное время выполнения парсинга + конвертации в цикле для трёх
методов"""
    print(f'{"Обязательное":^12}: {timeit.timeit("main_task()", "from __main__
import main_task", number=100)} сек.')
    print(f'{"1-е доп.":^12}: {timeit.timeit("extra_task_1()", "from __main__
import extra_task_1", number=100)} сек.')
    print(f'{"2-е доп.":^12}: {timeit.timeit("extra_task_2()", "from __main__
import extra_task_2", number=100)} сек.')

if __name__ == '__main__':
    extra_task_3()
```

Вывод программы:

```
Обязательное: 0.0348881550016813 сек.
  1-е доп.  : 0.10650223600168829 сек.
  2-е доп.  : 0.04222728999957326 сек.
```

Выходит, что самая быстрая реализация – конвертация вручную без регулярных выражений. Можно предположить, что рекурсивный алгоритм, который в ней реализован, оптимален для небольшого файла, а также не требует дополнительных обращений к библиотекам, таким как *re*.

Дополнительное задание № 4

Будем конвертировать исходный JSON-файл в формат PROTOBUF. Этот формат является усовершенствованной версией XML, позволяя уменьшить размер сериализованных данных и эффективность взаимодействия с ними в разы, по сравнению с обычным XML. Причём данные сериализуются в специальный байт-код, который невозможно прочитать вручную, в отличие от XML. Нужно использовать парсинг и расшифровку байтов из PROTOBUF.

Так как данный формат разработан компанией Google, то он поставляется с готовой библиотекой для сериализации и парсинга. Изначально нужно вручную описать структуру сериализуемых объектов и записать их в отдельный файл. В данном случае это файл *schedule.proto*.

schedule.proto

```
syntax = "proto2";

message Schedule {
    optional string day = 1;

    message Subject {
        optional string subjectName = 1;
        optional string beginTime = 2;
        optional string endTime = 3;
        repeated int32 weekNumbers = 4;
        optional string auditorium = 5;
        optional string place = 6;
        optional bool isEvenWeek = 7;
        optional string teacher = 8;
        optional string format = 9;
    }

    repeated Subject subjects = 2;
}
```

Затем необходимо скомпилировать этот файл в Python-модуль с помощью команды:

```
protoc -I=. --python_out=. ./schedule.proto
```

В директории появится файл *schedule_pb2.py*. Его можно импортировать и обращаться к готовым методам создания объекта, его сериализации и парсинга следующим образом:

main.py

```
import schedule_pb2
```

```
def main_task():
      # Её реализация приведена в обязательном задании

def extra_task_1():
      # Её реализация приведена в доп. задании 1

def extra_task_2():
      # Её реализация приведена в доп. задании 2

def extra_task_3():
      # Её реализация приведена в доп. задании 3

def extra_task_4():
    """Ковертировать schedule.json в protocol buffer"""
    # Сериализация
    data = parse_json_file('schedule.json')
    schedule_proto = schedule_pb2.Schedule()
    schedule_proto.day = data['day']
    for subject in data['subjects']:
        subject_proto = schedule_proto.subjects.add()
        for key, value in subject.items():
            if isinstance(value, list):
                getattr(subject_proto, key).extend(value)
            else:
                setattr(subject_proto, key, value)

    serialized_proto = schedule_proto.SerializeToString()
    with open('schedule_protobuf', 'wb') as f:
        f.write(serialized_proto)

    # Чтение
    with open('schedule_protobuf', 'rb') as f:
        data = f.read()

    new_schedule_proto = schedule_pb2.Schedule()
    new_schedule_proto.ParseFromString(data)
    print(f'День: {new_schedule_proto.day}')
    for subject_proto in new_schedule_proto.subjects:
        print(f'Предмет: {subject_proto.subjectName}')
        print(f'Время начала: {subject_proto.beginTime}')
        print(f'Время окончания: {subject_proto.beginTime}')
        print(f'Время окончания: {subject_proto.endTime}')
        print(f'Номера недель: {subject_proto.weekNumbers}')
        print(f'Аудитория: {subject_proto.auditorium}')
        print(f'Место проведения: {subject_proto.place}')
        print(f'Чётная неделя: {"да" if subject_proto.isEvenWeek else "нет"}')
        print(f'Чётная неделя: {"да" if subject_proto.isEvenWeek else "нет"}')
        print(f'Преподаватель: {subject_proto.teacher}')
        print(f'Формат проведения: {subject_proto.format}')
        print('-'*50)


if __name__ == '__main__':
    extra_task_4()
```

После выполнения блока с сериализацией считанных данных из JSON-файла в директории появится файл *schedule_protobuf*. Его содержимое:

schedule_protobuf (Рисунок 1)

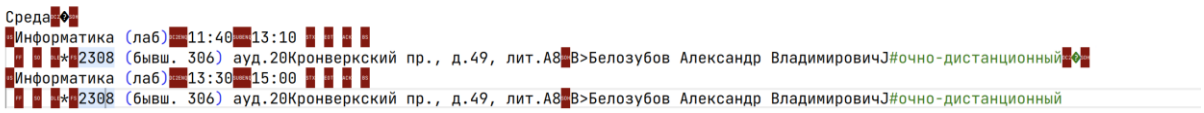

Рисунок 1

Как видно, его содержимое трудно прочитать и понять, что оно из себя представляет. Но формат PROTOBUF и не подразумевает, что сам байт-код будет кто-то читать. Данные из него нужно извлекать программно, что делает блок чтения в *main.py*. Он выведет прочитает байты из *schedule_protobuf*, выполнит их парсинг и выведет в читаемом формате в консоль:

Вывод main.py

```
День: Среда
Предмет: Информатика (лаб)
Время начала: 11:40
Время окончания: 11:40
Время окончания: 13:10
Номера недель: [2, 4, 6, 8, 10, 12, 14, 16]
Аудитория: 2308 (бывш. 306) ауд.
Место проведения: Кронверкский пр., д.49, лит.А
Чётная неделя: да
Чётная неделя: да
Преподаватель: Белозубов Александр Владимирович
Формат проведения: очно-дистанционный
--------------------------------------------------
Предмет: Информатика (лаб)
Время начала: 13:30
Время окончания: 13:30
Время окончания: 15:00
Номера недель: [2, 4, 6, 8, 10, 12, 14, 16]
Аудитория: 2308 (бывш. 306) ауд.
Место проведения: Кронверкский пр., д.49, лит.А
Чётная неделя: да
Чётная неделя: да
Преподаватель: Белозубов Александр Владимирович
Формат проведения: очно-дистанционный
--------------------------------------------------
```

Заключение

В результате выполнения данной лабораторной работы были изучены следующие вещи:

- Форматы JSON, YAML и PROTOBUF;
- Понятие парсинга данных и его практическая реализация для считывания JSON-файла;
- Понятие сериализации данных и её практическая реализация для записи данных в YAML-файл;
- Библиотеки для работы с JSON- и YAML-файлами;
- Применение регулярных выражений при парсинге файла.

Список использованной литературы

1. **Лямин А. В. и Череповская Е. Н.** Объектно-ориентированное программирование. Компьютерный практикум. [Книга]. - СПб : Университет ИТМО, 2017.
2. **Орлов С. А. Цилькер Б. Я.** Организация ЭВМ и систем: Учебник для вузов, 2-е издание [Книга]. - СПб : Питер, 2011.
3. **Салуев Тигран** Пишем изящный парсер на Питоне [В Интернете] // Хабр. - 5 Сентябрь 2016 г.. - https://habr.com/ru/post/309242/.
4. Форма Бэкуса-Наура [В Интернете] // Википедия. - https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0_%D0%91%D1%8D%D0%BA%D1%83%D1%81%D0%B0_%E2%80%94_%D0%9D%D0%B0%D1%83%D1%80%D0%B0.