Национальный Исследовательский Университет ИТМО
Факультет Программной Инженерии и Компьютерной Техники

Вариант № 17
Лабораторная работа № 4
по дисциплине
'Информатика'

Выполнил:
Студент группы P3113
***Куперштейн Дмитрий;* : 269359**
Преподаватель:
Малышева Татьяна Алексеевна

Санкт-Петербург 2019 г.

Содержание

1 Задание

1. Изучить форму Бэкуса-Наура.
2. Изучить особенности языков разметки/форматов JSON, YAML, XML, PROTOBUF
3. Понять устройство страницы с расписанием для своей группы:
   `http://www.ifmo.ru/ru/schedule/0/P3200/schedule.htm`
4. Исходя из структуры расписания конкретного дня, сформировать файл с расписанием в формате, указанном в задании в качестве исходного.
5. Написать программу на языке Python 3.x, которая бы осуществляла парсинг и конвертацию исходного файла в новый.
6. Нельзя использовать готовые библиотеки, кроме re (регулярные выражения в Python) и библиотеки для загрузки XML-файлов.
7. Необязательное задание для получения оценки «4» и «5» (позволяет набрать от 75 до 89 процентов от максимального числа баллов БаРС за данную лабораторную):
   (a) Найти готовые библиотеки, осуществляющие аналогичный парсинг и конвертацию файлов.
   (b) Сравнить полученные результаты и объяснить их сходство/различие.
8. Необязательное задание для получения оценки «5» (позволяет набрать от 90 до 100 процентов от максимального числа баллов БаРС за данную лабораторную):
   (a) Используя свою программу и найденные готовые библиотеки, сравнить десятикратное время выполнения парсинга + конвертации в цикле.
   (b) Проанализировать полученные результаты и объяснить их сходство/различие.
9. Проверить, что все пункты задания выполнены и выполнены верно.
10. Написать отчёт о проделанной работе.
11. Подготовиться к устным вопросам на защите.

Согласно варианту 17 необходимо распимание на среду в формате XML конвертировать в YAML.

2 Исходный фаил

2.1 schedule.xml

```
<?xml version="1.0" encoding="utf-8"?>
<schedule>
  <day num="wed">
      <event type="lec">
        <start>10:00</start>
        <stop>11:30</stop>
        <class>2219</class>
            <subject>math</subject>
            <level>base</level>
        <building>lomo</building>
        <teacher>Холодова Светлана Евгеньевна</teacher>
      </event>
      <event type="prac">
        <start>11:40</start>
        <stop>13:10</stop>
        <class></class>
            <subject>foreign languages</subject>
            <level></level>
        <building>lomo</building>
        <teacher></teacher>
      </event>
      <event type="prac">
        <start>15:20</start>
        <stop>16:50</stop>
        <class>1122</class>
            <subject>math</subject>
            <level>base</level>
        <building>lomo</building>
        <teacher>Попов Антон Игоревич</teacher>
      </event>
  </day>
</schedule>
```

3 Результат

3.1 schedule.yaml

```yaml
day:
  attributes:
    num: wed
  event:
    attributes:
      type: lec
    start: 10:00
    stop: 11:30
    class: 2219
    subject: math
    level: base
    building: lomo
    teacher: Холодова Светлана Евгеньевна
  event1:
    attributes:
      type: prac
    start: 11:40
    stop: 13:10
    class:
    subject: foreign languages
    level:
    building: lomo
    teacher:
  event2:
    attributes:
      type: prac
    start: 15:20
    stop: 16:50
    class: 1122
    subject: math
    level: base
    building: lomo
    teacher: Попов Антон Игоревич
```

4 Исходный код программы

4.1 inf_lab4.py

```
import xml_parser

root = xml_parser.parse('schedule.xml')
with open('schedule.yaml', 'w') as f:
    xml_parser.to_yaml(root, f)
```

4.2 xml_lexer.py

```
import re
from typing import List
from operator import xor

tag_re = re.compile(
    r"""^(?!<[xX][mM][lL])
    (<\s*([a-zA-Z_][-a-zA-Z_.\d]*)
    ((\s*[a-zA-Z_][-a-zA-Z_.\d]*=(['\"]).*\5)*)
    \s*(/)?>|</\s*([a-zA-Z_][-a-zA-Z_.\d]*)\s*>)""",
    re.X)

open_tag_re = re.compile(
    r"""^(?!<[xX][mM][lL])
    <\s*(?P<name>[a-zA-Z_][-a-zA-Z_.\d]*)
    (?P<attrs>(\s*[a-zA-Z_][-a-zA-Z_.\d]*=(['\"]).*\4)*)\s*>""",
    re.X)

close_tag_re = re.compile(
    r"""^(?!<[xX][mM][lL])</(?P<name>[a-zA-Z_][-a-zA-Z_.\d]*)\s*>""",
    re.X)

self_closed_tag_re = re.compile(r"""^(?!<[xX][mM][lL])
<\s*(?P<name>[a-zA-Z_][-a-zA-Z_.\d]*)
(?P<attrs>(\s*[a-zA-Z_][-a-zA-Z_.\d]*=(['\"]).*\4)*)\s*/>""",
                                re.X)

data_re = re.compile(r'\s*(?P<data>\S[^<>]+)')

attribute_re = re.compile(
    r"""[a-zA-Z_][-a-zA-Z_.\d]*=(["']).*\1""")

decl_re = re.compile(
    r"""<\?xml\s+
    version=['"](?P<ver>\d\.\d)['"]
    (?:\s+encoding=['"](?P<enc>[-a-zA-Z\d]+)['"])?
    (?:\s+standalone=['"](?P<stand>yes|no)['"])?
    \s*\?>""", re.X)


class XmlLexerError(Exception):
    pass
```

```
def read_xml_file(file: str) -> str:
    with open(file) as f:
        s = f.read().replace('\n', '').lstrip().rstrip()
    return s


def get_tokens(file: str) -> List[str]:
    tokens: List[str] = []
    s = read_xml_file(file)
    tmp_p = 0
    for p in range(len(s)):
        c = s[p]
        if c == '<':
            pros_data_match = data_re.match(s[tmp_p:p])
            if pros_data_match and pros_data_match.group('data'):
                tokens.append(pros_data_match.group('data'))
            tmp_p = p
        elif c == '>':
            pros_tag_match = tag_re.match(s[tmp_p:p + 1])
            pros_decl_match = decl_re.match(s[tmp_p:p + 1])
            if xor(bool(pros_tag_match), bool(pros_decl_match)):
                match = pros_tag_match or pros_decl_match
                tokens.append(
                    match.group(0))
            else:
                raise XmlLexerError(
                    'Invalid tag: {}'.format(s[tmp_p:p + 1]))
            tmp_p = p + 1
    return tokens
```

4.3 xml_parse.py

```
import re
from collections import Counter
from typing import List
from typing import NamedTuple
from typing import Optional
from typing import Union

import xml_lexer

XmlNode = Union['XmlSection', 'XmlElement']


class XmlParserError(Exception):
    pass


class XmlSection(NamedTuple):
    name: str
```

```
    attributes: dict
    parent: Optional['XmlSection']
    inc: List[XmlNode]


class XmlElement(NamedTuple):
    name: str
    value: str
    attributes: dict
    parent: XmlSection


def get_tag_attributes(tag_match: re.Match) -> dict:
    attributes = {}
    attrs_string = tag_match.group('attrs')
    if attrs_string is not None:
        for s in attrs_string.split():
            if not xml_lexer.attribute_re.match(s):
                raise XmlParserError(
                    'Wrong attribute {} in tag {}'.format(
                        s, tag_match.group(2)))
            name, value = s.split('=')
            value = value[1:-1]
            if name not in attributes:
                attributes[name] = value
            else:
                raise XmlParserError(
                    '>=2 attrs with same name in {} tag'.format(
                        tag_match.group(2)))
    return attributes


def __recur_parse(sec: XmlSection, tokens: List[str],
                  p: int = 0, stack=None) -> None:
    if stack is None:
        stack = []

    tag = tokens[p]
    pros_open = xml_lexer.open_tag_re.match(tag)
    if pros_open:
        if p + 2 <= len(tokens) - 1:
            pros_data = xml_lexer.data_re.match(tokens[p + 1])
            pros_close = xml_lexer.close_tag_re.match(tokens[p + 2])
            if pros_data and pros_close \
                    and pros_open.group('name') \
                    == pros_close.group('name'):
                sec.inc.append(
                    XmlElement(pros_open.group('name'),
                               pros_data.group('data'),
                               get_tag_attributes(pros_open),
                               sec))
                p += 3
```

```
                __recur_parse(sec, tokens, p, stack)
                return
        if p + 1 <= len(tokens) - 1:
            pros_close = xml_lexer.close_tag_re.match(tokens[p + 1])
            if pros_close and pros_close.group('name') \
                    == pros_open.group('name'):
                sec.inc.append(
                    XmlElement(pros_open.group('name'),
                               '',
                               get_tag_attributes(pros_open),
                               sec))
                p += 2
                __recur_parse(sec, tokens, p, stack)
                return
        stack.append(pros_open.group('name'))
        sec.inc.append(XmlSection(pros_open.group('name'),
                                  get_tag_attributes(pros_open),
                                  sec, []))
        sec = sec.inc[-1]
        p += 1
        __recur_parse(sec, tokens, p, stack)
        return
    pros_self_closed = xml_lexer.self_closed_tag_re.match(tag)
    if pros_self_closed:
        sec.inc.append(
            XmlElement(pros_self_closed.group('name'),
                       '',
                       get_tag_attributes(pros_self_closed),
                       sec))
    pros_close = xml_lexer.close_tag_re.match(tag)
    if pros_close:
        if stack.pop() != pros_close.group('name'):
            raise XmlParserError()
        sec = sec.parent
        if len(stack) == 0:
            return
        p += 1
        __recur_parse(sec, tokens, p, stack)


def __write_attrs_to_yaml(f, attrs: dict, d: int, indent='  ') -> None:
    f.write('{}{}:\n'.format(indent * d, 'attributes'))
    d += 1
    for name in attrs:
        f.write('{}{}: {}\n'.format(indent * d, name,
                                    attrs[name]))


def to_yaml(el: XmlSection, f, d=0, indent='  '):
    ex_name = Counter()
    for i in el.inc:
        name = i.name
```

```
            ex_name[name] += 1
            if ex_name[name] != 1:
                name += str(ex_name[i.name] - 1)

            if isinstance(i, XmlElement):
                f.write('{}{}:'.format(indent * d, name))
                if len(i.attributes) != 0:
                    f.write('\n')
                    d += 1
                    __write_attrs_to_yaml(f, i.attributes, d)
                    f.write('{}value: {}\n'.format(indent * d, i.value))
                else:
                    f.write(' {}\n'.format(i.value))
            elif isinstance(i, XmlSection):
                f.write('{}{}:\n'.format(indent * d, name))
                d += 1
                if len(i.attributes) != 0:
                    __write_attrs_to_yaml(f, i.attributes, d)
                to_yaml(i, f, d)
                d -= 1


def parse(file: str) -> Optional[XmlSection]:
    tokens = xml_lexer.get_tokens(file)
    if not len(tokens):
        return
    start = 0
    decl_match = xml_lexer.decl_re.match(tokens[0])
    if decl_match:
        version = '1.0' or decl_match.group('ver')
        encoding = 'utf-8' or decl_match.group('enc').lower()
        standalone = 'yes' or decl_match.group('stand')
        if version != '1.0' or encoding != 'utf-8' \
                or standalone != 'yes':
            raise XmlParserError(
                'Parse only standalone 1.0 xml in utf-8 encode')
        start = 1
    root_tag = tokens[start]
    root_tag_match = xml_lexer.open_tag_re.match(root_tag)
    if not root_tag_match:
        raise XmlParserError('No open root tag')
    root_attrs = get_tag_attributes(root_tag_match)
    root = XmlSection(root_tag_match.group('name'),
                      root_attrs, None, [])
    __recur_parse(root, tokens, start + 1, [root_tag_match.group('name')])
    return root
```

5 Вывод

В ходе этой лабораторной работы я реализовал парсинг XML файлов через абстрактое дерево и вывод дерева в YAML формате. Так же я изучил форму Бэкуса-Наура и особенности многочисленных языков разметки.