

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 №13

Выполнил

Лабин Макар Андреевич

группа Р3131

Проверила

Авксентьева Елена Юрьевна

к.п.н., доцент

г. Санкт-Петербург, 2024

Содержание

Задание.....	3
Основные этапы вычисления.....	5
Заключение.....	27
Список использованных источников.....	28

Задание

1. Исходя из структуры расписания конкретного дня, сформировать файл с расписанием в формате, указанном в задании в качестве исходного. При этом необходимо, чтобы хотя бы в одной из выбранных дней было не менее двух занятий (можно использовать своё персональное). В случае, если в данный день недели нет таких занятий, то увеличить номер варианта ещё на восемь.

2. Обязательное задание (позволяет набрать до 45 процентов от максимального числа баллов БаРС за данную лабораторную): написать программу на языке Python 3.x или любом другом, которая бы осуществляла парсинг и конвертацию исходного файла в новый путём простой замены метасимволов исходного формата на метасимволы результирующего формата.

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

4. Дополнительное задание №1 (позволяет набрать +10 процентов от максимального числа баллов БаРС за данную лабораторную).

а) Найти готовые библиотеки, осуществляющие аналогичный парсинг и конвертацию файлов.

б) Переписать исходный код, применив найденные библиотеки. Регулярные выражения также нельзя использовать.

в) Сравнить полученные результаты и объяснить их сходство/различие. Объяснение должно быть отражено в отчёте.

5. Дополнительное задание №2 (позволяет набрать +10 процентов от максимального числа баллов БаРС за данную лабораторную).

а) Переписать исходный код, добавив в него использование регулярных выражений.

б) Сравнить полученные результаты и объяснить их сходство/различие. Объяснение должно быть отражено в отчёте.

6. Дополнительное задание №3 (позволяет набрать +25 процентов от максимального числа баллов БаРС за данную лабораторную).

а) Переписать исходный код таким образом, чтобы для решения задачи использовались формальные грамматики. То есть ваш код должен уметь осуществлять парсинг и конвертацию любых данных, представленных в исходном формате, в данные, представленные в результирующем формате: как с готовыми библиотеками из дополнительного задания №1.

б) Проверку осуществить как минимум для расписания с двумя учебными днями по два занятия в каждом.

с) Сравнить полученные результаты и объяснить их сходство/различие. Объяснение должно быть отражено в отчёте.

7. Дополнительное задание №4 (позволяет набрать +5 процентов от максимального числа баллов БаРС за данную лабораторную).

а) Используя свою исходную программу из обязательного задания и программы из дополнительных заданий, сравнить стократное время выполнения парсинга + конвертации в цикле.

б) Проанализировать полученные результаты и объяснить их сходство/различие. Объяснение должно быть отражено в отчёте.

8. Дополнительное задание №5 (позволяет набрать +5 процентов от максимального числа баллов БаРС за данную лабораторную).

а) 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.

б) Проанализировать полученные результаты, объяснить особенности использования формата. Объяснение должно быть отражено в отчёте.

Основные этапы вычисления

Для варианта №13 имеем требования, которые представлены в Таблица 1.

Таблица 1 — Выдержка из текста лабораторной работы

№ варианта	Исходный формат	Результирующий формат	Дни недели
13	XML	JSON	Среда, суббота

Исходный файл будет сформирован исходя из расписания, представленного на Рисунок 1.

4, Ср	5, Чт	6, Пт	7, Сб
08:20 Информатика Балашин Павел Валерьевич ауд. Актовый зал (1216/0 (усл)) ул.Ломоносова, д.9, лит. М			08:20 Линейная алгебра Свинцов Михаил Викторович
10:00 Основы профессиональной деятельности Клименков Сергей Викторович ауд. Актовый зал (1216/0 (усл)) ул.Ломоносова, д.9, лит. М			
13:30 Программирование Бобрус Александр Владимирович ауд. 1415 Кронверкский пр., д.49, лит.А			13:30 Основы профессиональной деятельности Обляшевский Севастьян Александрович ауд. 2308 Кронверкский пр., д.49, лит.А
15:20 Программирование Бобрус Александр Владимирович ауд. 1415 Кронверкский пр., д.49, лит.А			15:20 Основы профессиональной деятельности Обляшевский Севастьян Александрович ауд. 2308 Кронверкский пр., д.49, лит.А

Составим исходный файл в формате XML:

```
<schedule>
  <weekday>
    <name>CP</name>
    <item>
      <time>08:20-09:50</time>
      <group>Информ 1.16 (P3131)</group>
      <teacher>Балашкин Павел Валерьевич</teacher>
      <room>Ауд. Актовый зал (1216/0 (усл)), ул.Ломоносова, д.9,
лит. М</room>
      <lesson>Информатика (лекция)</lesson>
      <lesson-format>Очно</lesson-format>
    </item>
    <item>
      <time>10:00-11:30</time>
      <group>Осн Проф деят 1.21 (P3131)</group>
      <teacher>Клименков Сергей Викторович</teacher>
```

```

        <room>Ауд. Актный зал (1216/0 (усл)), ул.Ломоносова, д.9,
лит. М</room>
        <lesson>Основы профессиональной деятельности (лекция)</lesson>
        <lesson-format>Очно</lesson-format>
    </item>
    <item>
        <time>13:30-15:00</time>
        <group>Прог 1.21 (P3131)</group>
        <teacher>Наумова Надежда Александровна</teacher>
        <room>Ауд. 1415, Кронверкский пр., д.49, лит.А</room>
        <lesson>Программирование (лабораторная)</lesson>
        <lesson-format>Очно</lesson-format>
    </item>
    <item>
        <time>15:20-16:50</time>
        <group>Прог 1.21 (P3131)</group>
        <teacher>Наумова Надежда Александровна</teacher>
        <room>Ауд. 1415, Кронверкский пр., д.49, лит.А</room>
        <lesson>Программирование (лабораторная)</lesson>
        <lesson-format>Очно</lesson-format>
    </item>
</weekday>
<weekday>
    <name>СБ</name>
    <item>
        <time>08:20-09:50</time>
        <group>Лин Алг 5.1</group>
        <teacher>Свинцов Михаил Викторович</teacher>
        <lesson>Линейная алгебра (лекция)</lesson>
        <lesson-format>Дистанционно</lesson-format>
    </item>
    <item>
        <time>13:30-15:00</time>
        <group>Осн Проф деят 1.21 (P3131)</group>
        <teacher>Обляшевский Севастьян Александрович</teacher>
        <room>Ауд. 2112, Кронверкский пр., д.49, лит.А</room>
        <lesson>Основы профессиональной деятельности
(лабораторная)</lesson>
        <lesson-format>Очно</lesson-format>
    </item>
    <item>
        <time>15:20-16:50</time>
        <group>Осн Проф деят 1.21 (P3131)</group>
        <teacher>Обляшевский Севастьян Александрович</teacher>
        <room>Ауд. 2112, Кронверкский пр., д.49, лит.А</room>
        <lesson>Основы профессиональной деятельности
(лабораторная)</lesson>
        <lesson-format>Очно</lesson-format>
    </item>
</weekday>
</schedule>

```

Для решения **обязательного задания** без использования регулярных выражений и готовых библиотек было написано два модуля: один для парсинга исходного файла в формате XML, другой для конвертации полученных данных в файл формата JSON. Оба модуля используют в качестве шаблона для унификации содержимое модуля Parser (тоже реализованного самостоятельно). В модуле XML происходит посимвольное сканирование

исходного файла и, в случае соответствия содержимого какому-либо метасимволу, в нужном контексте происходит считывание содержательной части и её запись в соответствующие переменные. Модуль JSON преобразует полученный после парсинга словарь в свой формат путём добавления соответствующих метасимволов в нужные места, получая из этого конечную строку, записываемую в отдельный файл. Для кооперации работы отдельных модулей программы был создан отдельный файл *task1.py*, являющийся точкой входа в программу.

Модуль Parser.py:

```
class Parser:
    def __init__(self, content=None, object=None, autogen=True):
        """
        An abstract parser class for markup languages.
        """
        self._content, self._object = content, object._object if
        isinstance(object, Parser) else object
        self._emptyContent = '<emptyContent>'
        self._tabSymbol = ' '
        self._tabCount = 3
        if autogen:
            self.autogenerate()

    def __str__(self):
        """
        Returns the string equivalent of the input Python
        dictionary.
        """
        return self._emptyContent if self._content is None else
        self._content

    def autogenerate(self):
        """
        Generates the omitting object/string of a parsed file.
        """
        if self._content is None:
            self.stringify_object()
        else:
            self.parse_string()
```

Модуль XML1.py:

```
from modules.Parser import *

class XML(Parser):
    def __init__(self, content=None, object=None, autogen=True):
        """
        A class which works with XML formatting.
        """
        super().__init__(content, object, False)
        self._emptyContent = '<emptyXML>'
        self.S = ' \n\r\t'
        self.screened = {'lt': '<', 'gt': '>', 'amp': '&', 'apos': '\'',
            'quot': '\"'}
        if autogen:
```

```

        self.autogenerate()

def parse_opening_tag(self, idx=0):
    """
    Returns the tuple (name, dict_of_metatags, is_closed)
    """
    tags, values, current, is_closed = list(), list(), '', 0
    idx += 1
    while True:
        match self._content[idx]:
            case '=': # assigning value to a metatag
                tags.append(current) ; current = ''
                values.append(self.parse_quotes(idx + 1))
                idx += 1
            case ' ': # splitting metatags
                tags.append(current) ; current = ''
            case '>': # closing tag
                if self._content[idx - 1] == '/':
                    tags.append(current[:-1])
                    is_closed = 1
                else:
                    tags.append(current)
                    break
            case _: # fill current keyword
                current += self._content[idx]
                idx += 1
    return (tags[0], dict(zip(tags[1:], values)), is_closed, idx + 1)

def add_tag_to_obj(self, obj, key, value):
    """
    Adds a key-value pair to an object in an XML way.
    """
    if isinstance(value, (dict, list)) and not len(value):
        value = None
    if key in obj:
        if isinstance(obj[key], list):
            obj[key].append(value)
        else:
            obj[key] = [obj[key], value]
    else:
        obj[key] = value
    return obj

def parse_closing_key(self, idx=0):
    """
    Returns the index of the next symbol just after the end of
    closing key.
    """
    while self._content[idx] != '>':
        idx += 1
    return idx + 1

def scan_until_symbol(self, idx=0):
    """
    Scans the content until the non-space symbol is encountered.
    """
    while self._content[idx] in ' \r\n\t':
        idx += 1
    return idx

```



```

def parse_screened(self, idx=0):
    """
    Parses the screened version of some chars.
    """
    code = ''
    while self._content[idx] != ';':
        code += self._content[idx]
        idx += 1
    return self.screened[code[1:]], idx

def parse_char(self, idx=0):
    """
    Parses the following char.
    """
    if self._content[idx] == '&':
        return self.parse_screened(idx)
    return self._content[idx], idx

def parse_tag(self, idx=0):
    """
    Parses the content of XML tag entirely and returns the tuple
    (tag_name, python_obj, idx).
    """
    fields = dict()
    while not self.is_key(True, idx):
        idx = self.scan_until_symbol(idx)
        if self.is_comment(idx):
            idx = self.parse_comment(idx)
        name, meta, is_closed, idx = self.parse_opening_tag(idx)
        for key in meta:
            fields['_' + key] = meta[key]
        if not is_closed:
            idx = self.scan_until_symbol(idx)
            while not self.is_key(False, idx):
                if self.is_key(True, idx): # if the following is an opening
tag
                    if isinstance(fields, str):
                        fields = {'__text': fields}
                    inner_name, obj, idx = self.parse_tag(idx)
                    fields = self.add_tag_to_obj(fields, inner_name, obj)
                else: # if the following is the tag's value (or a comment)
                    if self.is_comment(idx):
                        idx = self.parse_comment(idx)
                        continue
                    char, idx = self.parse_char(idx)
                    if isinstance(fields, dict):
                        if fields == dict() and char not in self.S:
                            fields = char
                        else:
                            if '__text' in fields:
                                fields['__text'] += char
                            elif self._content[idx] not in self.S:
                                fields['__text'] = char
                    else:
                        fields += char
                    idx += 1
            idx = self.parse_closing_key(idx)
        if isinstance(fields, dict) and '__text' in fields:
            fields['__text'] = repr(fields['__text'].rstrip())[1:-1]
        elif isinstance(fields, str):

```

```

        fields = repr(fields.rstrip())[1:-1]
    return (name, fields, idx)

def parse_quotes(self, idx=0):
    """
    Reads the content of quoted string 'as is'.
    """
    result, quote = '', self._content[idx]
    idx += 1
    while self._content[idx] != quote:
        result += self._content[idx]
        idx += 1
    return result

def is_comment(self, idx=0):
    """
    Checks if the string is a comment.
    """
    return self._content[idx:idx+2] == '<!'

def parse_comment(self, idx=0):
    """
    Omits the whole comment.
    """
    while self._content[idx-2:idx+1] != '-->':
        idx += 1
    return idx + 1

def is_key(self, opening=False, idx=0):
    """
    Checks if the following symbol is tag (opening/closing).
    """
    if self._content[idx:idx+2] == '<!':
        return False
    return self._content[idx] == '<' and self._content[idx + 1] !=
'/' if opening else self._content[idx:idx+2] == '</'

def parse_string(self):
    """
    Parses the content of input XML string into object.
    """
    name, obj, _ = self.parse_tag()
    self._object = {name: obj}

```

Модуль JSON.py:

```

from modules.Parser import *

class JSON(Parser):
    def __init__(self, content=None, object=None, autogen=True):
        """
        A class which works with JSON formatting.
        """
        super().__init__(content, object, False)
        self._emptyContent = '<emptyJSON>'
        if autogen:
            self.autogenerate()

    def format_primitives(self, item):
        """
        Formats primitives in a JSON-like manner.

```

```

    """
    if item is None:
        return 'null'
    if not isinstance(item, str):
        return item
    return '"' + item.replace("\'", "\\\'") + '"'

def add_indentations(self, depth):
    return '\n' + self._tabSymbol * self._tabCount * depth

def parse_structure(self, object, depth=0):
    """
    Parses the tag structure and returns its string equivalent.
    """
    if not isinstance(object, (dict, list)):
        return self.format_primitives(object)
    else:
        if isinstance(object, dict):
            string_result, keys = '{' + self.add_indentations(depth +
1), list(object.keys())
            N = len(keys)
            for i in range(N):
                string_result += self.format_primitives(str(keys[i]))
                string_result += ': '
                string_result += self.parse_structure(object[keys[i]],
depth + 1)
                if i != N - 1:
                    string_result += ',' + self.add_indentations(depth +
1)
            string_result += self.add_indentations(depth) + '}'
        else:
            string_result, N = '[' + self.add_indentations(depth + 1),
len(object)
            for i in range(N):
                string_result += self.parse_structure(object[i], depth +
1)
                if i != N - 1:
                    string_result += ',' + self.add_indentations(depth +
1)
            string_result += self.add_indentations(depth) + ']'
        return string_result

def stringify_object(self):
    """
    Fills the str content of JSON object from given object.
    """
    self._content = self.parse_structure(self._object)

```

Файл task1.py:

```

from modules.XML1 import *
from modules.JSON import *

def main():
    """
    Main function.
    """
    with open('schedule.xml') as f:
        with open('schedule.out', 'w') as g:
            xml = XML(content=f.read())
            json = JSON(object=xml)

```

```

g.write(str(json))

if __name__ == '__main__':
    main()

```

В итоге был получен следующий файл:

```

{
  "schedule": {
    "weekday": [
      {
        "name": "СР",
        "item": [
          {
            "time": "08:20-09:50",
            "group": "Информ 1.16 (Р3131)",
            "teacher": "Балакшин Павел Валерьевич",
            "room": "Ауд. Актовый зал (1216/0 (усл)), ул.Ломоносова, д.9, лит. М",
            "lesson": "Информатика (лекция)",
            "lesson-format": "Очно"
          },
          {
            "time": "10:00-11:30",
            "group": "Осн Проф деят 1.21 (Р3131)",
            "teacher": "Клименков Сергей Викторович",
            "room": "Ауд. Актовый зал (1216/0 (усл)), ул.Ломоносова, д.9, лит. М",
            "lesson": "Основы профессиональной деятельности (лекция)",
            "lesson-format": "Очно"
          },
          {
            "time": "13:30-15:00",
            "group": "Прог 1.21 (Р3131)",
            "teacher": "Наумова Надежда Александровна",
            "room": "Ауд. 1415, Кронверкский пр., д.49, лит.А",
            "lesson": "Программирование (лабораторная)",
            "lesson-format": "Очно"
          },
          {
            "time": "15:20-16:50",
            "group": "Прог 1.21 (Р3131)",
            "teacher": "Наумова Надежда Александровна",
            "room": "Ауд. 1415, Кронверкский пр., д.49, лит.А",
            "lesson": "Программирование (лабораторная)",
            "lesson-format": "Очно"
          }
        ]
      },
      {
        "name": "СБ",
        "item": [
          {
            "time": "08:20-09:50",
            "group": "Лин Алг 5.1",
            "teacher": "Свинцов Михаил Викторович",
            "lesson": "Линейная алгебра (лекция)",
            "lesson-format": "Дистанционно"
          },
          {

```

```

        "time": "13:30-15:00",
        "group": "Осн Проф деят 1.21 (Р3131)",
        "teacher": "Обляшевский Севастьян Александрович",
        "room": "Ауд. 2112, Кронверкский пр., д.49, лит.А",
        "lesson": "Основы профессиональной деятельности
(лабораторная)",
        "lesson-format": "Очно"
    },
    {
        "time": "15:20-16:50",
        "group": "Осн Проф деят 1.21 (Р3131)",
        "teacher": "Обляшевский Севастьян Александрович",
        "room": "Ауд. 2112, Кронверкский пр., д.49, лит.А",
        "lesson": "Основы профессиональной деятельности
(лабораторная)",
        "lesson-format": "Очно"
    }
]
}
]
}
}

```

Для выполнения **дополнительного задания №1** были найдены следующие готовые библиотеки:

- *xmlltodict*, производящая парсинг файла XML в словарь;
- *json* (стандартная библиотека), производящая ковертацию словаря в JSON-строку.

В файле *task2.py* они были последовательно применены к исходному файлу.

Файл task2.py:

```
import xmltodict as XML
import json as JSON

def main():
    """
    Main function.
    """
    with open('schedule.xml') as f:
        with open('schedule.out', 'w') as g:
            xml = XML.parse(f.read())
            JSON.dump(xml, g, ensure_ascii=False, indent=3)

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

Выходной результат такой же, как и в первом задании. Это объясняется тем, что в исходном файле отсутствуют атрибуты у тегов и другие особенности XML, которые можно было бы интерпретировать по-разному при конвертации в JSON из-за отсутствия аналогичных метасимволов в формате JSON.

Для выполнения **дополнительного задания №2** был переписан модуль XML с использованием регулярных выражений. Различие в коде состоит в том, что вместо посимвольного сканирования строки происходит проверка частей строки на соответствие некоторым шаблонам, которые объединяют наборы метасимволов. В модуле JSON не

происходит сканирования строки, поэтому применение регулярных выражений не имеет смысла в нём. Поэтому в данном задании нет разницы между применением собственного модуля конвертации или стороннего. В дальнейшем выбор модуля конвертации будет сделан в пользу стороннего решения.

Файл XML3.py:

```
import re

from modules.Parser import *

class XML(Parser):
    def __init__(self, content=None, object=None, autogen=True):
        """
        A class which works with XML formatting.
        """
        super().__init__(content, object, False)
        self._emptyContent = '<emptyXML>'
        # patterns
        self.tag_pattern = re.compile(r'<([\w-]+)(?:([.]*)(\w*))?>')
        self.attr_pattern = re.compile(r'([\w-]+)=(\'|")(.*)\2')
        self.opening_tag_pattern = re.compile(r'<\w')
        self.closing_tag_pattern = re.compile(r'<\/[\w-]+>')
        self.value_pattern = re.compile(r'[\n\r\t]*([<&]+)')
        self.screened_pattern = re.compile(r'&(\w+);')
        self.comment_pattern = re.compile(r'<!--.*?-->')
        self.S = re.compile(r'[\r\n\t]+')
        self.screened = {'lt': '<', 'gt': '>', 'amp': '&', 'apos': '\'',
            'quot': '\"'}
        if autogen:
            self.autogenerate()

    def parse_opening_tag(self, idx=0):
        """
        Returns the tuple (name, dict_of_metatags, is_closed)
        """
        attrs, values = list(), list()
        tag_obj = self.tag_pattern.search(self._content, idx)
        if tag_obj.group(2) is not None:
            for attr in self.attr_pattern.finditer(tag_obj.group(2)):
                attrs.append(attr.group(1))
                values.append(attr.group(3))
            return (tag_obj.group(1), dict(zip(attrs, values)),
tag_obj.group(3) is not None, tag_obj.end())

    def add_tag_to_obj(self, obj, key, value):
        """
        Adds a key-value pair to an object in an XML way.
        """
        if isinstance(value, (dict, list)) and not len(value):
            value = None
        if key in obj:
            if isinstance(obj[key], list):
                obj[key].append(value)
            else:
                obj[key] = [obj[key], value]
        else:
            obj[key] = value
        return obj
```

```

def add_string_to_obj(self, obj, string):
    if isinstance(obj, dict):
        if obj == dict() and not self.S.fullmatch(string):
            obj = string.lstrip()
        else:
            if '__text' in obj:
                obj['__text'] += string
            elif not self.S.fullmatch(string):
                obj['__text'] = string.lstrip()
    else:
        obj += string
    return obj

def parse_screened(self, code):
    """
    Parses the screened version of some chars.
    """
    return self.screened[code]

def parse_tag(self, idx=0):
    """
    Parses the content of XML tag entirely and returns the tuple
    (tag_name, python_obj, idx).
    """
    fields = dict()
    name, meta, is_closed, idx = self.parse_opening_tag(idx)
    for key in meta:
        fields['_ ' + key] = meta[key]
    if not is_closed:
        while not self.closing_tag_pattern.match(self._content, idx):
            # if the following is a tag
            if self.opening_tag_pattern.match(self._content, idx):
                if isinstance(fields, str):
                    fields = {'__text': fields}
                inner_name, obj, idx = self.parse_tag(idx)
                fields = self.add_tag_to_obj(fields, inner_name, obj)
            # if the following is a comment
            elif self.comment_pattern.match(self._content, idx):
                idx = self.comment_pattern.match(self._content,
idx).end()
            # if the following is the tag's value
            elif self.value_pattern.match(self._content, idx):
                value_obj = self.value_pattern.search(self._content,
idx)
                fields = self.add_string_to_obj(fields,
value_obj.group(0))
                idx = value_obj.end()
            # if the following is a screened symbol
            elif self.screened_pattern.match(self._content, idx):
                screened_obj =
self.screened_pattern.search(self._content, idx)
                fields = self.add_string_to_obj(fields,
self.parse_screened(screened_obj.group(1)))
                idx = screened_obj.end()
                if idx >= len(self._content):
                    break
            else:
                idx = self.closing_tag_pattern.match(self._content,
idx).end()

```

```

    if isinstance(fields, dict) and '__text' in fields:
        fields['__text'] = fields['__text'].rstrip()
    elif isinstance(fields, str):
        fields = fields.rstrip()
    return (name, fields, idx)

```

```

def parse_string(self):
    """
    Parses the content of input XML string into object.
    """
    name, obj, _ = self.parse_tag()
    self._object = {name: obj}

```

Файл task3.py:

```

from modules.XML3 import *
import json as JSON

def main():
    """
    Main function.
    """
    with open('schedule.xml') as f:
        with open('schedule.out', 'w') as g:
            xml = XML(content=f.read())
            JSON.dump(xml._object, g, ensure_ascii=False, indent=3)

if __name__ == '__main__':
    main()

```

Выходной результат такой же, как и в первом задании.

Для выполнения **дополнительного задания №3** был изучен набор формальных грамматик XML, который можно изучить на сайте w3.org¹. Так как их количество составляет 90 шт., в модуле парсинга XML были реализованы лишь те из них, которые позволяют конвертировать исходный файл в формат JSON. В качестве примера было взято решение из рекомендованного списка литературы к лабораторной работе².

Файл XML4.py:

```

import re

from modules.Parser import *

def sequence(*funcs):
    """
    Yields tuple if all of funcs are matched sequentially.
    """
    if len(funcs) == 0:
        def result(idx=0):
            yield (), idx
            return result
    def result(idx=0):
        for arg1, idx in funcs[0](idx):
            for others, idx in sequence(*funcs[1:])(idx):
                yield (arg1,) + others, idx
    return result

def lor(*funcs):
    """

```



```

Yields the first non-None match.
"""
def result(idy=0):
    for func in funcs:
        for val, idy in func(idy):
            yield val, idy
        return
    return result

def question(func):
    """
    Yields the match once if possible.
    """
    def result(idy=0):
        for val, idy in func(idy):
            yield val, idy
        return
    yield None, idy
    return result

def asterisk(func):
    """
    Yields the list of func as much as possible.
    """
    def result(idy=0):
        for (val1, others), idy in sequence(func,
                                             asterisk(func))(idy):
            yield [val1] + others, idy
        return
    yield [], idy
    return result

class XML(Parser):
    class Pattern:
        Char = re.compile('[\t\r\n\u0020-\u007F\uE000-\uFFFF\u00010000-\u0010FFFF]')
        S = re.compile('[\t\r\n]+')
        Eq = re.compile('[\t\r\n]*=[\t\r\n]*')
        NameStartChar = '[:A-Z_a-z\u00C0-\u00D6\u00D8-\u00F6\u00F8-\u02FF\u037D\u037F-\u1FFF\u200C-\u200D\u2070-\u218F\u2C00-\u2FEF\u3001-\u07FF\uF900-\uFDCF\uFDF0-\uFFFF\u00010000-\u000EFFFF]'
        NameChar = '(:' + NameStartChar + '|[-.0-9\u00B7\u0300-\u036F\u203F-\u2040])'
        Name = re.compile(NameStartChar + NameChar + '*')
        AttValue = re.compile('"([^\&"]*)"')
        CharData = re.compile('[^\&]+')
        EntityRef = {'lt': '<', 'gt': '>', 'amp': '&', 'apos': '\'', 'quot': '\"'}

    def __init__(self, content=None, object=None, autogen=True):
        """
        A class which works with XML formatting.
        """
        super().__init__(content, object, False)
        self._emptyContent = '<emptyXML>'
        if autogen:
            self.autogenerate()

    def minus(self, pattern, chars):
        """

```

```

Subtracts 'chars' from matched pattern symbol.
"""
def result(idx=0):
    obj = pattern.match(self._content[idx:])
    if obj is not None:
        if obj.group(0) not in chars:
            yield obj.group(0), idx + obj.end()
    return result

def parseCharData(self):
    def result(idx=0):
        obj = self.Pattern.CharData.match(self._content[idx:])
        if obj is not None:
            yield (obj.group(0), idx + obj.end())
    return result

def parseWord(self, word, value=None):
    def result(idx=0):
        if self._content[idx:].startswith(word):
            yield value, idx + len(word)
    return result

def parseS(self):
    def result(idx=0):
        obj = self.Pattern.S.match(self._content[idx:])
        if obj is not None:
            yield (None, idx + obj.end())
    return result

def parseName(self):
    def result(idx=0):
        obj = self.Pattern.Name.match(self._content[idx:])
        if obj is not None:
            yield (obj.group(0), idx + obj.end())
    return result

def parseEq(self):
    def result(idx=0):
        obj = self.Pattern.Eq.match(self._content[idx:])
        if obj is not None:
            yield (None, idx + obj.end())
    return result

def parseAttValue(self):
    def result(idx=0):
        obj = self.Pattern.AttValue.match(self._content[idx:])
        if obj is not None:
            yield (obj.group(1), idx + obj.end())
    return result

def parseAttribute(self):
    def result(idx=0):
        for (name, _, attValue), idx in sequence(self.parseName(),
                                                  self.parseEq(),
                                                  self.parseAttValue())
(id):
        yield {name: attValue}, idx
    return result

def parseEmptyElemTag(self):

```

```

        def result(idx=0):
            for (_, name, keyvalues, _, _), idx in
sequence(self.parseWord('<'),
                                                self.parseName
()),
                                                asterisk(seque
nce(self.parseS(),
self.parseAttribute()))),
                                                question(self.
parseS()),
                                                self.parseWord
('>'))(idx):
                tag = {name: dict()}
                for kv in keyvalues:
                    if kv is not None:
                        _, kv = kv
                        for k in kv.keys():
                            tag[name]['_' + k] = kv[k]
                if not len(tag[name]):
                    tag[name] = None
                yield tag, idx
            return result

        def parseSTag(self):
            def result(idx=0):
                for (_, name, keyvalues, _, _), idx in
sequence(self.parseWord('<'),
                                                self.parseName
()),
                                                asterisk(seque
nce(self.parseS(),
self.parseAttribute()))),
                                                question(self.
parseS()),
                                                self.parseWord
('>'))(idx):
                    tag = {name: dict()}
                    for kv in keyvalues:
                        if kv is not None:
                            _, kv = kv
                            for k in kv.keys():
                                tag[name]['_' + k] = kv[k]
                    yield tag, idx
                return result

            def parseETag(self):
                def result(idx=0):
                    for (_, _, _, _), idx in sequence(self.parseWord('</'),
                                                        self.parseName(),
                                                        question(self.parseS()),
                                                        self.parseWord('>'))(idx):
                        yield None, idx
                    return result

            def parseComment(self):
                def result(idx=0):
                    for _, idx in sequence(self.parseWord('<!--'),

```

```

ar, '-'),
                                asterisk(lor(self.minus(self.Pattern.Ch
                                sequence(self.parseWord('-
', '-'),
                                self.minus(self.P
attern.Char, '-')))),
                                self.parseWord('-->'))(idx):
        yield None, idx
        return result

    def parseElemTag(self):
        def result(idx=0):
            for (fields, content, _), idx in sequence(self.parseSTag(),
                                                    self.parseContent(),
                                                    self.parseETag())
(idx):
                key = next(iter(fields))
                if isinstance(content, str):
                    if len(fields[key]) == 0:
                        fields[key] = content
                    else:
                        fields[key]['__text'] = content
                else:
                    for subkey in content.keys():
                        self.add_tag_to_obj(fields[key], subkey,
content[subkey])
                    if not len(content):
                        fields[key] = None
                yield fields, idx
            return result

    def parseElement(self):
        def result(idx=0):
            for value, idx in lor(self.parseEmptyElemTag(),
                                self.parseElemTag())(idx):
                yield value, idx
            return result

    def parseEntityRef(self):
        def result(idx=0):
            for (_, name, _), idx in sequence(self.parseWord('&'),
                                                    self.parseName(),
                                                    self.parseWord(';'))(idx):
                yield self.Pattern.EntityRef[name], idx
            return result

    def parseContent(self):
        def result(idx=0):
            fields = dict()
            for (value, other), idx in
sequence(question(self.parseCharData()),
                                asterisk(sequence(lor(s
elf.parseElement(),
                                                    s
elf.parseEntityRef(),
                                                    s
elf.parseComment()),
                                quest
ion(self.parseCharData()))))(idx):
                if value is not None and value.lstrip():

```

```

        fields = value
    for var, data in other:
        if var is not None:
            if isinstance(var, str):
                if isinstance(fields, dict):
                    fields = var
                else:
                    fields += var
            else:
                if isinstance(fields, str):
                    fields = {'__text': fields}
                for key in var:
                    self.add_tag_to_obj(fields, key, var[key])
        if data is not None and data.lstrip():
            if isinstance(fields, dict):
                fields['__text'] = fields.get('__text', '') + data
            else:
                fields += data
    yield fields, idx
return result

def add_tag_to_obj(self, obj, key, value):
    """
    Adds a key-value pair to an object in an XML way.
    """
    if key in obj:
        if isinstance(obj[key], list):
            obj[key].append(value)
        else:
            obj[key] = [obj[key], value]
    else:
        obj[key] = value
    return obj

def parse_string(self):
    """
    Parses the content of input XML string into object.
    """
    for value, _ in self.parseElement():
        self._object = value

```

Файл task4.py:

```

from modules.XML4 import *
import json as JSON

def main():
    """
    Main function.
    """
    with open('schedule.xml') as f:
        with open('schedule.out', 'w') as g:
            xml = XML(content=f.read())
            JSON.dump(xml._object, g, ensure_ascii=False, indent=3)

if __name__ == '__main__':
    main()

```

Выходной результат такой же, как и в первом задании.

Для выполнения дополнительного задания №4 была написана программа *task5.py*, считающая процессорное время стократного исполнения парсинга исходного файла и его конвертации в JSON формат во всех предыдущих заданиях.

Файл task5.py:

```
import time

from modules.XML1 import XML as XML1
import xmltodict as XML2
from modules.XML3 import XML as XML3
from modules.XML4 import XML as XML4
from modules.JSON import *
import json as JSON2

def xml1(content):
    xml = XML1(content=content)
    json = JSON(object=xml)

def xml2(content):
    xml = XML2.parse(content)
    JSON2.dumps(xml, ensure_ascii=False, indent=3)

def xml3(content):
    xml = XML3(content=content)
    JSON2.dumps(xml._object, ensure_ascii=False, indent=3)

def xml4(content):
    xml = XML4(content=content)
    JSON2.dumps(xml._object, ensure_ascii=False, indent=3)

def measure_ptime(func, *args, COUNT=100):
    """
    Measures the process time of COUNT executions of an input funcion.
    """
    start = time.process_time()
    for _ in range(COUNT):
        func(*args)
    stop = time.process_time()
    return f'{stop - start:.4f}'

def main():
    """
    Main function.
    """
    with open('schedule.xml') as f:
        content = f.read()
    print('task1.py:', measure_ptime(xml1, content))
    print('task2.py:', measure_ptime(xml2, content))
    print('task3.py:', measure_ptime(xml3, content))
    print('task4.py:', measure_ptime(xml4, content))

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

В результате были получены данные, представленные на Рисунок 2.

```
task1.py: 0.1683
task2.py: 0.0321
task3.py: 0.0559
task4.py: 0.5821
```

Рисунок 2

Из данных видно, что программа task4.py выполняется дольше всех. Это можно объяснить тем, что в коде используются рекурсивные и внутренние функции, которые замедляют анализ строки. Программа task2.py выполняется более чем в пять раз быстрее task1.py из-за использования встроенной библиотеки для работы с XML-файлами. Программа task3.py медленнее task2.py из-за особенностей регулярных выражений: на больших массивах данных регулярные выражения могут замедлять работу программы.

Для выполнения дополнительного задания №5 в качестве результирующего формата был выбран TOML (Tom's Obvious Minimal Language). Для этого был переписан модуль конвертации словаря, чтобы удовлетворять требованиям нового формата. Данный модуль не предполагает сканирование строки, поэтому в решении не использовались регулярные выражения. Выбор модуля парсинга в данном задании не важен, поэтому в файле *task6.py* используется сторонний модуль.

Файл TOML.py:

```
from modules.Parser import *

class TOML(Parser):
    def __init__(self, content=None, object=None, autogen=True):
        """
        A class which works with TOML formatting.
        """
        super().__init__(content, object, False)
        self._emptyContent = '<emptyTOML>'
        if autogen:
            self.autogenerate()

    def format_primitives(self, item):
        """
        Formats primitives in a TOML-like manner.
        """
        if not isinstance(item, str):
            return item
        return "'" + item.replace("\'", "\\'\") + "'"

    def is_dict_primitive(self, obj):
        """
        Check if the dict has inner dicts.
        """
        for key in obj.keys():
            if isinstance(obj[key], list):
                return self.is_list_primitive(obj[key])
            return not isinstance(obj[key], dict)

    def is_list_primitive(self, obj):
        """
        Checks if the list has inner dicts.
        """
        for key in obj:
```

```

        return not isinstance(key, (list, dict))

    def generate_tree(self, object, parent=''):
        result = list()
        if isinstance(object, dict):
            stash_items = list()
            object_items = list()
            for key in object.keys():
                if isinstance(object[key], list) and
self.is_list_primitive(object[key]):
                    stash_items.append((key, object[key]))
                elif isinstance(object[key], (list, dict)):
                    object_items += self.generate_tree(object[key], parent +
'.' + key if parent else key)
                else:
                    stash_items.append((key, object[key]))
            if stash_items:
                result.append((parent, stash_items))
            if object_items:
                result += object_items
        elif isinstance(object, list):
            if self.is_list_primitive(object):
                result.append((parent, object))
            else:
                for item in object:
                    result += self.generate_tree(item, parent)
        return result

    def get_array_tags(self, tree):
        """
        Returns the array dict tags.
        """
        visited, array = set(), set()
        for item in tree:
            if item[0] in visited:
                array.add(item[0])
            else:
                visited.add(item[0])
        return array

    def parse_structure(self, object):
        """
        Parses the tag structure and returns its string equivalent.
        """
        string_result = ''
        tree = self.generate_tree(object)
        array_tags = self.get_array_tags(tree)
        for tag, value in tree:
            if tag in array_tags:
                string_result += f'\n[{tag}]\n'
            else:
                string_result += f'\n{tag}\n'
            if isinstance(value, list):
                for name, info in value:
                    string_result += f'{name} =
{self.format_primitives(info)}\n'
            else:
                string_result += f'{tag[0]} =
{self.format_primitives(tag[1])}\n'
        return string_result

```



```
def stringify_object(self):
    """
    Fills the str content of TOML object from given object.
    """
    self._content = self.parse_structure(self._object).strip()
```

Файл task6.py:

```
from modules.XML1 import *
from modules.TOML import *

def main():
    """
    Main function.
    """
    with open('schedule.xml') as f:
        with open('schedule.out', 'w') as g:
            xml = XML(content=f.read())
            toml = TOML(object=xml)
            g.write(str(toml))

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

В итоге был получен следующий файл:

```
[[schedule.weekday]]
name = "СР"

[[schedule.weekday.item]]
time = "08:20-09:50"
group = "Информ 1.16 (Р3131)"
teacher = "Балакшин Павел Валерьевич"
room = "Ауд. Актовый зал (1216/0 (усл)), ул.Ломоносова, д.9, лит. М"
lesson = "Информатика (лекция)"
lesson-format = "Очно"

[[schedule.weekday.item]]
time = "10:00-11:30"
group = "Осн Проф деят 1.21 (Р3131)"
teacher = "Клименков Сергей Викторович"
room = "Ауд. Актовый зал (1216/0 (усл)), ул.Ломоносова, д.9, лит. М"
lesson = "Основы профессиональной деятельности (лекция)"
lesson-format = "Очно"

[[schedule.weekday.item]]
time = "13:30-15:00"
group = "Прог 1.21 (Р3131)"
teacher = "Наумова Надежда Александровна"
room = "Ауд. 1415, Кронверкский пр., д.49, лит.А"
lesson = "Программирование (лабораторная)"
lesson-format = "Очно"

[[schedule.weekday.item]]
time = "15:20-16:50"
group = "Прог 1.21 (Р3131)"
teacher = "Наумова Надежда Александровна"
room = "Ауд. 1415, Кронверкский пр., д.49, лит.А"
lesson = "Программирование (лабораторная)"
lesson-format = "Очно"
```

```

[[schedule.weekday]]
name = "СБ"

[[schedule.weekday.item]]
time = "08:20-09:50"
group = "Лин Алг 5.1"
teacher = "Свинцов Михаил Викторович"
lesson = "Линейная алгебра (лекция)"
lesson-format = "Дистанционно"

[[schedule.weekday.item]]
time = "13:30-15:00"
group = "Осн Проф деят 1.21 (Р3131)"
teacher = "Обляшевский Севастьян Александрович"
room = "Ауд. 2112, Кронверкский пр., д.49, лит.А"
lesson = "Основы профессиональной деятельности (лабораторная)"
lesson-format = "Очно"

[[schedule.weekday.item]]
time = "15:20-16:50"
group = "Осн Проф деят 1.21 (Р3131)"
teacher = "Обляшевский Севастьян Александрович"
room = "Ауд. 2112, Кронверкский пр., д.49, лит.А"
lesson = "Основы профессиональной деятельности (лабораторная)"
lesson-format = "Очно"

```

Выбранный формат полученного файла похож по своей внешней простоте на JSON или YAML, однако он также позволяет однозначно конвертировать исходный файл в свой формат без потери содержательной информации. Стоит отметить, что TOML не поддерживает массивы с разными типами данных, поэтому не все данные можно конвертировать из XML в TOML

Заключение

В ходе выполнения лабораторной работы была изучена структура различных Markup языков, в частности XML, JSON и TOML. Основываясь на этих знаниях, был составлен файл с учебным расписанием двух дней недели, написанный в формате XML. Для выполнения всех дополнительных заданий нужно было работать с формальными грамматиками XML и уметь читать форму Бэкуса-Наура. Несмотря на одинаковый результат парсинга и конвертации исходного файла с расписанием, были проанализированы сходства и различия написанных программ, которые отражены в отчёте.

Список использованных источников

1. Extensible Markup Language (XML) 1.0 (Fifth Edition) // World Wide Web Consortium [Электронный ресурс]. Режим доступа: <https://www.w3.org/TR/REC-xml/> (дата обращения 20.11.2024).

2. Т. Салуев. Пишем изящный парсер на Питоне // «Хабр» [Электронный ресурс]. Режим доступа: <https://habr.com/ru/post/309242/> (дата обращения 20.11.2024).