Министерство высшего образования и науки Российской Федерации

Национальный научно-исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №4

по дисциплине

**«Информатика»**

Вариант № 367837 % 36 = 25 (Пт.) –(+8)-> 33 (Сб.) –(+8 %36)-> 5 (Пн.) –(+8)-> 13 (Ср.)

Работу выполнил:

Афанасьев Кирилл Александрович,

Студент группы P3106

Преподаватель:

Балакшин Павел Валерьевич

Санкт-Петербург, 2023

Оглавление

[Задание 3](#_Toc134701949)

[Основные этапы вычисления 4](#_Toc134701950)

[Исходный файл: 4](#_Toc134701951)

[Обязательное задание: 5](#_Toc134701952)

[Исходный код: 5](#_Toc134701953)

[Содержимое результирующего файла: 8](#_Toc134701954)

[Дополнительное задание 1: 9](#_Toc134701955)

[Исходный код: 9](#_Toc134701956)

[Содержимое результирующего файла: 10](#_Toc134701957)

[Микровывод: 11](#_Toc134701958)

[Дополнительное задание 2: 11](#_Toc134701959)

[Исходный код: 11](#_Toc134701960)

[Содержимое результирующего файла: 13](#_Toc134701961)

[Микровывод: 14](#_Toc134701962)

[Дополнительное задание 3: 14](#_Toc134701963)

[Код программы тестирования: 14](#_Toc134701964)

[Результаты тестирования: 15](#_Toc134701965)

[Микровывод: 15](#_Toc134701966)

[Дополнительное задание 4: 15](#_Toc134701967)

[Исходный код: 16](#_Toc134701968)

[Содержимое результирующего файла: 19](#_Toc134701969)

[Микровывод: 20](#_Toc134701970)

[Вывод 20](#_Toc134701971)

[Список литературы 20](#_Toc134701972)

# Задание

1. Исходя из структуры расписания конкретного дня, сформировать файл с расписанием в формате, указанном в задании в качестве исходного.
2. Обязательное задание:
   1. Написать программу на языке Python 3.x, которая бы осуществляла парсинг и конвертацию исходного файла в новый. Нельзя использовать готовые библиотеки, в том числе регулярные выражения в Python и библиотеки для загрузки XML-файлов.
3. 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.
4. Дополнительное задание 2:
   1. Переписать исходный код, добавив в него использование регулярных выражений.
   2. Сравнить полученные результаты и объяснить их сходство/различие.
5. Дополнительное задание 3:
   1. Используя свою исходную программу из обязательного задания, программу из дополнительного задания No1 и программу из дополнительного задания No2, сравнить стократное время выполнения парсинга + конвертации в цикле.
   2. Проанализировать полученные результаты и объяснить их сходство/различие.
6. 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.

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант 13: | XML | JSON | Среда |

# Основные этапы вычисления

Исходный код всего парсера (основной скрипт + 4 доп. скриптов), со всеми приведенными далее примерами файлов: <https://github.com/Zerumi/inf_labs/tree/master/mylabs/lab4>

Используем расписание, доступное по ссылке: <https://itmo.ru/ru/schedule/0/P3106/schedule.htm>

## Исходный файл:

input.txt:

<schedule>

<group>P3106</group>

<semester>Spring</semester>

<day dayname="Среда">

<subject type="Лабораторная">

<time>13:30-15:00</time>

<week>четная</week>

<name>Программирование</name>

<teacher>Байрамова Хумай Бахруз Кызы</teacher>

<place>Ауд. 1338 (Бывш. 381), Кронверкский пр., д.49, лит.А</place>

<weeks>2, 4, 6, 8, 10, 12, 14, 16</weeks>

<mode>Очно - дистанционный</mode>

</subject>

<subject type="Лабораторная">

<time>13:30-15:00</time>

<week>нечетная</week>

<name>Основы профессиональной деятельности</name>

<teacher>Афанасьев Дмитрий Борисович</teacher>

<place>Ауд. 2435/5 (Бывш. 431Г), Кронверкский пр., д.49, лит.А</place>

<weeks>1, 3, 7, 11, 13, 15, 17, 18</weeks>

<mode>Очный</mode>

</subject>

<subject type="Лабораторная">

<time>15:20-16:50</time>

<week>четная</week>

<name>Программирование</name>

<teacher>Байрамова Хумай Бахруз Кызы</teacher>

<place>Ауд. 1338 (Бывш. 381), Кронверкский пр., д.49, лит.А</place>

<weeks>2, 4, 6, 8, 10, 12, 14, 16</weeks>

<mode>Очно - дистанционный</mode>

</subject>

<subject type="Лабораторная">

<time>15:20-16:50</time>

<week>нечетная</week>

<name>Основы профессиональной деятельности</name>

<teacher>Афанасьев Дмитрий Борисович</teacher>

<place>Ауд. 2435/5 (Бывш. 431Г), Кронверкский пр., д.49, лит.А</place>

<weeks>1, 3, 7, 11, 13, 15, 17, 18</weeks>

<mode>Очный</mode>

</subject>

</day>

</schedule>

## Обязательное задание:

### Исходный код:

main.py:

# я ненавижу пайтон  
# РЕВОЛЮЦИЯ В СФЕРЕ ПАРСИНГА!!!  
def parse\_xml(xml\_string):  
 *"""  
 Парсит XML-строку в коллекцию элементов.  
 """* # Список для хранения дочерних элементов  
 children = []  
  
 # Пока строка не пуста  
 while xml\_string:  
 # Удаляем пробелы и переносы строк в начале и конце строки  
 xml\_string = xml\_string.strip()  
  
 # Ищем первый открывающий тег  
 start\_tag\_start = xml\_string.find('<')  
 start\_tag\_end = xml\_string.find('>')  
  
 # Проверка на пролог  
 if xml\_string[start\_tag\_start:start\_tag\_start + 5] == '<?xml':  
 xml\_string = xml\_string[start\_tag\_end + 1:].strip()  
 continue  
  
 # Если нашли открывающий тег  
 if start\_tag\_start != -1 and start\_tag\_end != -1 and start\_tag\_start < start\_tag\_end:  
 # Ищем имя тега  
 tag\_name\_start = start\_tag\_start + 1  
 tag\_name\_end = xml\_string.find(' ', tag\_name\_start)  
 if tag\_name\_end == -1 or tag\_name\_end > start\_tag\_end:  
 tag\_name\_end = start\_tag\_end  
 tag\_name = xml\_string[tag\_name\_start:tag\_name\_end]  
  
 # Ищем закрывающий тег  
 end\_tag\_start = xml\_string.find('</' + tag\_name + '>', tag\_name\_start + 1)  
 test\_for\_duplicate = xml\_string.find('<' + tag\_name + '>', tag\_name\_start + 1)  
 while test\_for\_duplicate < end\_tag\_start and test\_for\_duplicate != -1:  
 end\_tag\_start = xml\_string.find('</' + tag\_name + '>', end\_tag\_start + 1)  
 test\_for\_duplicate = xml\_string.find('<' + tag\_name + '>', test\_for\_duplicate + 1)  
 end\_tag\_end = end\_tag\_start + len(tag\_name) + 3  
  
 # Ищем атрибуты тега  
 attributes = {}  
 attr\_start = tag\_name\_end  
 while True:  
 attr\_name\_start = xml\_string.find(' ', attr\_start, start\_tag\_end)  
 if attr\_name\_start == -1:  
 break  
 attr\_name\_end = xml\_string.find('=', attr\_name\_start)  
 if attr\_name\_end == -1:  
 break  
 attr\_name = xml\_string[attr\_name\_start:attr\_name\_end].strip()  
 attr\_value\_start = xml\_string.find('"', attr\_name\_end)  
 if attr\_value\_start == -1:  
 break  
 attr\_value\_end = xml\_string.find('"', attr\_value\_start + 1)  
 if attr\_value\_end == -1:  
 break  
 attr\_value = xml\_string[attr\_value\_start + 1:attr\_value\_end]  
 attributes[attr\_name] = attr\_value  
 attr\_start = attr\_value\_end  
  
 # Ищем текст элемента  
 text = ''  
 text\_start = start\_tag\_end + 1  
 # Пропускаем пробелы  
 while xml\_string[text\_start].isspace():  
 text\_start += 1  
 if not xml\_string[text\_start] == '<':  
 text\_end = end\_tag\_start  
 text = xml\_string[text\_start:text\_end].strip()  
  
 # Рекурсивно парсим дочерние элементы  
 inner\_xml\_start = start\_tag\_end + 1  
 inner\_xml\_end = end\_tag\_start  
 inner\_xml = xml\_string[inner\_xml\_start:inner\_xml\_end]  
 children.append({  
 'name': tag\_name,  
 'attributes': attributes,  
 'text': text,  
 'children': parse\_xml(inner\_xml)  
 })  
  
 # Дополняем до массива, если можно  
 while end\_tag\_end < len(xml\_string):  
 # Обрезаем строку  
 xml\_string\_nextstart = xml\_string.find('<', end\_tag\_end + 1)  
 xml\_string\_nextstart\_end = xml\_string.find('>', xml\_string\_nextstart + 1)  
 # вычисляем конец тега  
 xml\_string\_nextstart\_nameend = xml\_string.find(' ', xml\_string\_nextstart + 1)  
 if xml\_string\_nextstart\_nameend == -1 or xml\_string\_nextstart\_nameend > xml\_string\_nextstart\_end:  
 xml\_string\_nextstart\_nameend = xml\_string\_nextstart\_end  
 tagname = xml\_string[xml\_string\_nextstart + 1:xml\_string\_nextstart\_nameend]  
 xml\_closetagname = xml\_string.find('</' + tagname + '>', xml\_string\_nextstart\_end + 1)  
 test\_for\_duplicate = xml\_string.find('<' + tagname + '>', xml\_string\_nextstart\_end + 1)  
 while test\_for\_duplicate < xml\_closetagname and test\_for\_duplicate != -1:  
 xml\_closetagname = xml\_string.find('</' + tagname + '>', xml\_closetagname + 1)  
 test\_for\_duplicate = xml\_string.find('<' + tagname + '>', test\_for\_duplicate + 1)  
 xml\_string\_nextstop = xml\_closetagname + len(tagname) + 3  
 # Проверяем, что это не окончание какого-то тега  
 if xml\_string[xml\_string\_nextstart + 1] == '/':  
 break  
 single\_xml = xml\_string[xml\_string\_nextstart:xml\_string\_nextstop]  
 parsed\_xml = parse\_xml(single\_xml)  
 if len(parsed\_xml) == 1:  
 children.append(parsed\_xml[0])  
 end\_tag\_end = xml\_string\_nextstop  
 return children  
  
 # Возвращаем список дочерних элементов  
 return children  
  
  
def find\_nth(haystack, needle, n):  
 start = haystack.find(needle)  
 while start >= 0 and n > 1:  
 start = haystack.find(needle, start + len(needle))  
 n -= 1  
 return start  
  
  
def to\_json\_str(elements\_xml, tab\_count=0, is\_first\_iter=True):  
 result = ''  
 is\_array = False  
 union = union\_by\_name(elements\_xml)  
 if len(union) < len(elements\_xml):  
 is\_array = True  
 if is\_first\_iter:  
 result += '{\n'  
 tab\_count += 1  
 if is\_array:  
 for union\_array\_index in union:  
 matched\_non\_single\_element = len(union[union\_array\_index]) > 1  
 if matched\_non\_single\_element:  
 result += '\t' \* tab\_count + '\"' + union\_array\_index + '\" : [\n'  
 tab\_count += 1  
 for element in union[union\_array\_index]:  
 if matched\_non\_single\_element:  
 result += '\t' \* tab\_count + '{\n'  
 tab\_count += 1  
 for atr in element['attributes']:  
 result += '\t' \* tab\_count + '\"' + atr + '\":\"' + element['attributes'][atr] + '\",\n'  
 if element['text']:  
 result += '\t' \* tab\_count + '\"' + element['name'] + '\":\"' + element['text'] + '\",\n'  
 if element['children']:  
 result += to\_json\_str(element['children'], tab\_count, False)  
 if matched\_non\_single\_element:  
 result = result.rstrip(',\n')  
 result += '\n'  
 tab\_count -= 1  
 result += '\t' \* tab\_count + '},\n'  
 if matched\_non\_single\_element:  
 result = result.rstrip(',\n')  
 result += '\n'  
 if len(union[union\_array\_index]) > 1:  
 tab\_count -= 1  
 result += '\t' \* tab\_count + '],\n'  
 result = result.rstrip(',\n')  
 result += '\n'  
 else:  
 for element in elements\_xml:  
 if len(element['children']) > 1:  
 result += '\t' \* tab\_count + '\"' + element['name'] + '\": {\n'  
 tab\_count += 1  
 for atr in element['attributes']:  
 result += '\t' \* tab\_count + '\"' + atr + '\":\"' + element['attributes'][atr] + '\",\n'  
 if element['text']:  
 result += '\t' \* tab\_count + '\"' + element['name'] + '\":\"' + element['text'] + '\",\n'  
 if element['children']:  
 result += to\_json\_str(element['children'], tab\_count, False)  
 if len(element['children']) > 1:  
 tab\_count -= 1  
 result += '\t' \* tab\_count + '},\n'  
 result = result.rstrip(',\n')  
 result += '\n'  
 if is\_first\_iter:  
 result += '}\n'  
 return result  
  
  
def union\_by\_name(list\_data):  
 result\_dict = {}  
  
 for value in list\_data:  
 name = value["name"]  
 if name in result\_dict:  
 result\_dict[name].append(value)  
 else:  
 result\_dict[name] = [value]  
  
 return result\_dict  
  
  
def main():  
 elements\_from\_xml = parse\_xml(open('input.xml').read())  
 # print(elements\_from\_xml)  
 open('output.json', mode='w').write(to\_json\_str(elements\_from\_xml))

### Содержимое результирующего файла:

{  
 "schedule": {  
 "group":"P3106",  
 "semester":"Spring",  
 "day": {  
 "dayname":"Среда",  
 "subject" : [  
 {  
 "type":"Лабораторная",  
 "time":"13:30-15:00",  
 "week":"четная",  
 "name":"Программирование",  
 "teacher":"Байрамова Хумай Бахруз Кызы",  
 "place":"Ауд. 1338 (Бывш. 381), Кронверкский пр., д.49, лит.А",  
 "weeks":"2, 4, 6, 8, 10, 12, 14, 16",  
 "mode":"Очно - дистанционный"  
 },  
 {  
 "type":"Лабораторная",  
 "time":"13:30-15:00",  
 "week":"нечетная",  
 "name":"Основы профессиональной деятельности",  
 "teacher":"Афанасьев Дмитрий Борисович",  
 "place":"Ауд. 2435/5 (Бывш. 431Г), Кронверкский пр., д.49, лит.А",  
 "weeks":"1, 3, 7, 11, 13, 15, 17, 18",  
 "mode":"Очный"  
 },  
 {  
 "type":"Лабораторная",  
 "time":"15:20-16:50",  
 "week":"четная",  
 "name":"Программирование",  
 "teacher":"Байрамова Хумай Бахруз Кызы",  
 "place":"Ауд. 1338 (Бывш. 381), Кронверкский пр., д.49, лит.А",  
 "weeks":"2, 4, 6, 8, 10, 12, 14, 16",  
 "mode":"Очно - дистанционный"  
 },  
 {  
 "type":"Лабораторная",  
 "time":"15:20-16:50",  
 "week":"нечетная",  
 "name":"Основы профессиональной деятельности",  
 "teacher":"Афанасьев Дмитрий Борисович",  
 "place":"Ауд. 2435/5 (Бывш. 431Г), Кронверкский пр., д.49, лит.А",  
 "weeks":"1, 3, 7, 11, 13, 15, 17, 18",  
 "mode":"Очный"  
 }  
 ]  
 }  
 }  
}

## Дополнительное задание 1:

Найденные библиотеки: xmltodict + json. Их комбинация чем-то может напомнить то, что попытался реализовать и я.

### Исходный код:

additional1.py:

# после этого мир кажется светлым и прекрасным  
# написано сразу после окончания написания ручного парсера  
import xmltodict  
import json  
  
  
def main():  
 xml = open('input.xml').read()  
 ordered\_dict = xmltodict.parse(xml)  
 json\_str = json.dumps(ordered\_dict, ensure\_ascii=False, indent=4)  
 open('output\_add1.json', 'w').write(json\_str)

### Содержимое результирующего файла:

output\_add1.json:

{  
 "schedule": {  
 "group": "P3106",  
 "semester": "Spring",  
 "day": {  
 "@dayname": "Среда",  
 "subject": [  
 {  
 "@type": "Лабораторная",  
 "time": "13:30-15:00",  
 "week": "четная",  
 "name": "Программирование",  
 "teacher": "Байрамова Хумай Бахруз Кызы",  
 "place": "Ауд. 1338 (Бывш. 381), Кронверкский пр., д.49, лит.А",  
 "weeks": "2, 4, 6, 8, 10, 12, 14, 16",  
 "mode": "Очно - дистанционный"  
 },  
 {  
 "@type": "Лабораторная",  
 "time": "13:30-15:00",  
 "week": "нечетная",  
 "name": "Основы профессиональной деятельности",  
 "teacher": "Афанасьев Дмитрий Борисович",  
 "place": "Ауд. 2435/5 (Бывш. 431Г), Кронверкский пр., д.49, лит.А",  
 "weeks": "1, 3, 7, 11, 13, 15, 17, 18",  
 "mode": "Очный"  
 },  
 {  
 "@type": "Лабораторная",  
 "time": "15:20-16:50",  
 "week": "четная",  
 "name": "Программирование",  
 "teacher": "Байрамова Хумай Бахруз Кызы",  
 "place": "Ауд. 1338 (Бывш. 381), Кронверкский пр., д.49, лит.А",  
 "weeks": "2, 4, 6, 8, 10, 12, 14, 16",  
 "mode": "Очно - дистанционный"  
 },  
 {  
 "@type": "Лабораторная",  
 "time": "15:20-16:50",  
 "week": "нечетная",  
 "name": "Основы профессиональной деятельности",  
 "teacher": "Афанасьев Дмитрий Борисович",  
 "place": "Ауд. 2435/5 (Бывш. 431Г), Кронверкский пр., д.49, лит.А",  
 "weeks": "1, 3, 7, 11, 13, 15, 17, 18",  
 "mode": "Очный"  
 }  
 ]  
 }  
 }  
}

### Микровывод:

Основное отличие данного вывода от основной программы: дополнительно бывшие атрибуты XML тегов помечаются знаком @, чего не делается в основной программе. Учитывая гибкость основной программы, это может быть легко исправлено, но для наглядности разницы это решено было не делать. Также в библиотечном парсере слегка улучшено форматирование JSON строки: есть пробел после каждого двоеточия. Все остальные части вывода совпадают.

## Дополнительное задание 2:

### Исходный код:

additional2.py:

import re  
  
  
def parse\_xml(xml\_string):  
 root\_pattern = re.compile(r'<(\w+)(.\*?)>(.\*?)</\1>', re.DOTALL)  
 attr\_pattern = re.compile(r'(\w+)="([^"]+)"')  
 xml\_string = xml\_string.strip()  
  
 node\_match = root\_pattern.match(xml\_string)  
 result = []  
 if node\_match:  
 tag\_name = node\_match.group(1)  
 attributes = dict(attr\_pattern.findall(node\_match.group(2)))  
 text = ''  
 children = parse\_xml(node\_match.group(3).strip())  
 if len(children) == 0:  
 text = node\_match.group(3)  
 result.append({'name': tag\_name, 'attributes': attributes, 'text': text, 'children': children})  
 cursor = node\_match.span()[1]  
 while cursor < len(xml\_string):  
 temp\_count = len(xml\_string[cursor::])  
 node\_match = root\_pattern.match(xml\_string[cursor::].lstrip())  
 if node\_match:  
 tag\_name = node\_match.group(1)  
 attributes = dict(attr\_pattern.findall(node\_match.group(2)))  
 text = ''  
 children = parse\_xml(node\_match.group(3).strip())  
 if len(children) == 0:  
 text = node\_match.group(3)  
 result.append({'name': tag\_name, 'attributes': attributes, 'text': text, 'children': children})  
 cursor += node\_match.span()[1] + temp\_count - len(xml\_string[cursor::].lstrip())  
 return result  
  
  
def to\_json\_str(elements\_xml, tab\_count=0, is\_first\_iter=True):  
 result = ''  
 is\_array = False  
 union = union\_by\_name(elements\_xml)  
 if len(union) < len(elements\_xml):  
 is\_array = True  
 if is\_first\_iter:  
 result += '{\n'  
 tab\_count += 1  
 if is\_array:  
 for union\_array\_index in union:  
 matched\_non\_single\_element = len(union[union\_array\_index]) > 1  
 if matched\_non\_single\_element:  
 result += '\t' \* tab\_count + '\"' + union\_array\_index + '\" : [\n'  
 tab\_count += 1  
 for element in union[union\_array\_index]:  
 if matched\_non\_single\_element:  
 result += '\t' \* tab\_count + '{\n'  
 tab\_count += 1  
 for atr in element['attributes']:  
 result += '\t' \* tab\_count + '\"' + atr + '\":\"' + element['attributes'][atr] + '\",\n'  
 if element['text']:  
 result += '\t' \* tab\_count + '\"' + element['name'] + '\":\"' + element['text'] + '\",\n'  
 if element['children']:  
 result += to\_json\_str(element['children'], tab\_count, False)  
 if matched\_non\_single\_element:  
 result = result.rstrip(',\n')  
 result += '\n'  
 tab\_count -= 1  
 result += '\t' \* tab\_count + '},\n'  
 if matched\_non\_single\_element:  
 result = result.rstrip(',\n')  
 result += '\n'  
 if len(union[union\_array\_index]) > 1:  
 tab\_count -= 1  
 result += '\t' \* tab\_count + '],\n'  
 result = result.rstrip(',\n')  
 result += '\n'  
 else:  
 for element in elements\_xml:  
 if len(element['children']) > 1:  
 result += '\t' \* tab\_count + '\"' + element['name'] + '\": {\n'  
 tab\_count += 1  
 for atr in element['attributes']:  
 result += '\t' \* tab\_count + '\"' + atr + '\":\"' + element['attributes'][atr] + '\",\n'  
 if element['text']:  
 result += '\t' \* tab\_count + '\"' + element['name'] + '\":\"' + element['text'] + '\",\n'  
 if element['children']:  
 result += to\_json\_str(element['children'], tab\_count, False)  
 if len(element['children']) > 1:  
 tab\_count -= 1  
 result += '\t' \* tab\_count + '},\n'  
 result = result.rstrip(',\n')  
 result += '\n'  
 if is\_first\_iter:  
 result += '}\n'  
 return result  
  
  
def union\_by\_name(list\_data):  
 result\_dict = {}  
  
 for value in list\_data:  
 name = value["name"]  
 if name in result\_dict:  
 result\_dict[name].append(value)  
 else:  
 result\_dict[name] = [value]  
  
 return result\_dict  
  
  
def main():  
 xml\_str = open('input.xml').read()  
 parsed\_xml = parse\_xml(xml\_str)  
 # print(parsed\_xml)  
 json\_string = to\_json\_str(parsed\_xml)  
 # print(json\_string)  
 open('output\_add2.json', 'w').write(json\_string)

### Содержимое результирующего файла:

output\_add2.json:

{  
 "schedule": {  
 "group":"P3106",  
 "semester":"Spring",  
 "day": {  
 "dayname":"Среда",  
 "subject" : [  
 {  
 "type":"Лабораторная",  
 "time":"13:30-15:00",  
 "week":"четная",  
 "name":"Программирование",  
 "teacher":"Байрамова Хумай Бахруз Кызы",  
 "place":"Ауд. 1338 (Бывш. 381), Кронверкский пр., д.49, лит.А",  
 "weeks":"2, 4, 6, 8, 10, 12, 14, 16",  
 "mode":"Очно - дистанционный"  
 },  
 {  
 "type":"Лабораторная",  
 "time":"13:30-15:00",  
 "week":"нечетная",  
 "name":"Основы профессиональной деятельности",  
 "teacher":"Афанасьев Дмитрий Борисович",  
 "place":"Ауд. 2435/5 (Бывш. 431Г), Кронверкский пр., д.49, лит.А",  
 "weeks":"1, 3, 7, 11, 13, 15, 17, 18",  
 "mode":"Очный"  
 },  
 {  
 "type":"Лабораторная",  
 "time":"15:20-16:50",  
 "week":"четная",  
 "name":"Программирование",  
 "teacher":"Байрамова Хумай Бахруз Кызы",  
 "place":"Ауд. 1338 (Бывш. 381), Кронверкский пр., д.49, лит.А",  
 "weeks":"2, 4, 6, 8, 10, 12, 14, 16",  
 "mode":"Очно - дистанционный"  
 },  
 {  
 "type":"Лабораторная",  
 "time":"15:20-16:50",  
 "week":"нечетная",  
 "name":"Основы профессиональной деятельности",  
 "teacher":"Афанасьев Дмитрий Борисович",  
 "place":"Ауд. 2435/5 (Бывш. 431Г), Кронверкский пр., д.49, лит.А",  
 "weeks":"1, 3, 7, 11, 13, 15, 17, 18",  
 "mode":"Очный"  
 }  
 ]  
 }  
 }  
}

### Микровывод:

Структура моего парсера представляет из себя нечто следующее (в данном дополнительном задании изменению подверглись выделенные фиолетовым части):

XML –(парсинг)-> Дерево объектов –(парсинг)-> JSON

Это очень гибкая структура в рамках данного задания. Ведь чтобы переписать код используя регулярные выражения, можно вообще не затрагивать ту часть, где идет преобразование в JSON. Этим объясняется абсолютное сходство выводов основной программы и данной.

## Дополнительное задание 3:

Проведем тестирование сделанных ранее программ. Исходный код программы тестирования приведен ниже, результаты тестов изложены в рисунке 1.

### Код программы тестирования:

additional3.py:

import time  
from main import main as main\_task  
from additional1 import main as additional1\_task  
from additional2 import main as additional2\_task  
  
  
def benchmark(function):  
 start\_time = time.time()  
 for \_ in range(100):  
 function()  
 print("--- %s seconds ---" % (time.time() - start\_time))  
  
  
print("> pure\_python...")  
benchmark(main\_task)  
  
print("> external\_library...")  
benchmark(additional1\_task)  
  
print("> regular\_expressions...")  
benchmark(additional2\_task)

### Результаты тестирования:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1. Результаты тестирования.

### Микровывод:

Как можно заметить, библиотечный парсер вырывается вперед. Однако мой парсер на чистом Python не сильно ему уступает. Результаты закономерны: многие вещи в моем парсере можно было написать эффективнее, например:

# Пропускаем пробелы  
while xml\_string[text\_start].isspace():  
 text\_start += 1

Данный код можно заменить на комбинацию функций lstrip() и len(), вычисляя разницу между исходной строкой и укороченной.

Самый отстающий участник забега: парсер на регулярных выражениях. Это и не удивительно, если знать о скорости работы регулярных выражений. Зато они более удобны в использовании.

## Дополнительное задание 4:

Для выходной разметки был выбран язык Markdown.

### Исходный код:

additional4.py:

# я... синдзи икари  
def parse\_xml(xml\_string):  
 *"""  
 Парсит XML-строку в коллекцию элементов.  
 """* # Список для хранения дочерних элементов  
 children = []  
  
 # Пока строка не пуста  
 while xml\_string:  
 # Удаляем пробелы и переносы строк в начале и конце строки  
 xml\_string = xml\_string.strip()  
  
 # Ищем первый открывающий тег  
 start\_tag\_start = xml\_string.find('<')  
 start\_tag\_end = xml\_string.find('>')  
  
 # Проверка на пролог  
 if xml\_string[start\_tag\_start:start\_tag\_start + 5] == '<?xml':  
 xml\_string = xml\_string[start\_tag\_end + 1:].strip()  
 continue  
  
 # Если нашли открывающий тег  
 if start\_tag\_start != -1 and start\_tag\_end != -1 and start\_tag\_start < start\_tag\_end:  
 # Ищем имя тега  
 tag\_name\_start = start\_tag\_start + 1  
 tag\_name\_end = xml\_string.find(' ', tag\_name\_start)  
 if tag\_name\_end == -1 or tag\_name\_end > start\_tag\_end:  
 tag\_name\_end = start\_tag\_end  
 tag\_name = xml\_string[tag\_name\_start:tag\_name\_end]  
  
 # Ищем закрывающий тег  
 end\_tag\_start = xml\_string.find('</' + tag\_name + '>', tag\_name\_start + 1)  
 test\_for\_duplicate = xml\_string.find('<' + tag\_name + '>', tag\_name\_start + 1)  
 while test\_for\_duplicate < end\_tag\_start and test\_for\_duplicate != -1:  
 end\_tag\_start = xml\_string.find('</' + tag\_name + '>', end\_tag\_start + 1)  
 test\_for\_duplicate = xml\_string.find('<' + tag\_name + '>', test\_for\_duplicate + 1)  
 end\_tag\_end = end\_tag\_start + len(tag\_name) + 3  
  
 # Ищем атрибуты тега  
 attributes = {}  
 attr\_start = tag\_name\_end  
 while True:  
 attr\_name\_start = xml\_string.find(' ', attr\_start, start\_tag\_end)  
 if attr\_name\_start == -1:  
 break  
 attr\_name\_end = xml\_string.find('=', attr\_name\_start)  
 if attr\_name\_end == -1:  
 break  
 attr\_name = xml\_string[attr\_name\_start:attr\_name\_end].strip()  
 attr\_value\_start = xml\_string.find('"', attr\_name\_end)  
 if attr\_value\_start == -1:  
 break  
 attr\_value\_end = xml\_string.find('"', attr\_value\_start + 1)  
 if attr\_value\_end == -1:  
 break  
 attr\_value = xml\_string[attr\_value\_start + 1:attr\_value\_end]  
 attributes[attr\_name] = attr\_value  
 attr\_start = attr\_value\_end  
  
 # Ищем текст элемента  
 text = ''  
 text\_start = start\_tag\_end + 1  
 # Пропускаем пробелы  
 while xml\_string[text\_start].isspace():  
 text\_start += 1  
 if not xml\_string[text\_start] == '<':  
 text\_end = end\_tag\_start  
 text = xml\_string[text\_start:text\_end].strip()  
  
 # Рекурсивно парсим дочерние элементы  
 inner\_xml\_start = start\_tag\_end + 1  
 inner\_xml\_end = end\_tag\_start  
 inner\_xml = xml\_string[inner\_xml\_start:inner\_xml\_end]  
 children.append({  
 'name': tag\_name,  
 'attributes': attributes,  
 'text': text,  
 'children': parse\_xml(inner\_xml)  
 })  
  
 # Дополняем до массива, если можно  
 while end\_tag\_end < len(xml\_string):  
 # Обрезаем строку  
 xml\_string\_nextstart = xml\_string.find('<', end\_tag\_end + 1)  
 xml\_string\_nextstart\_end = xml\_string.find('>', xml\_string\_nextstart + 1)  
 # вычисляем конец тега  
 xml\_string\_nextstart\_nameend = xml\_string.find(' ', xml\_string\_nextstart + 1)  
 if xml\_string\_nextstart\_nameend == -1 or xml\_string\_nextstart\_nameend > xml\_string\_nextstart\_end:  
 xml\_string\_nextstart\_nameend = xml\_string\_nextstart\_end  
 tagname = xml\_string[xml\_string\_nextstart + 1:xml\_string\_nextstart\_nameend]  
 xml\_closetagname = xml\_string.find('</' + tagname + '>', xml\_string\_nextstart\_end + 1)  
 test\_for\_duplicate = xml\_string.find('<' + tagname + '>', xml\_string\_nextstart\_end + 1)  
 while test\_for\_duplicate < xml\_closetagname and test\_for\_duplicate != -1:  
 xml\_closetagname = xml\_string.find('</' + tagname + '>', xml\_closetagname + 1)  
 test\_for\_duplicate = xml\_string.find('<' + tagname + '>', test\_for\_duplicate + 1)  
 xml\_string\_nextstop = xml\_closetagname + len(tagname) + 3  
 # Проверяем, что это не окончание какого-то тега  
 if xml\_string[xml\_string\_nextstart + 1] == '/':  
 break  
 single\_xml = xml\_string[xml\_string\_nextstart:xml\_string\_nextstop]  
 parsed\_xml = parse\_xml(single\_xml)  
 if len(parsed\_xml) == 1:  
 children.append(parsed\_xml[0])  
 end\_tag\_end = xml\_string\_nextstop  
 return children  
  
 # Возвращаем список дочерних элементов  
 return children  
  
  
def find\_nth(haystack, needle, n):  
 start = haystack.find(needle)  
 while start >= 0 and n > 1:  
 start = haystack.find(needle, start + len(needle))  
 n -= 1  
 return start  
  
  
def to\_markdown\_str(elements\_xml, current\_head\_count=1):  
 if current\_head\_count > 6:  
 current\_head\_count = 6  
 result = ''  
 is\_array = False  
 union = union\_by\_name(elements\_xml)  
 if len(union) < len(elements\_xml):  
 is\_array = True  
 if is\_array:  
 for union\_array\_index in union:  
 matched\_non\_single\_element = len(union[union\_array\_index]) > 1  
 index = 1  
 for element in union[union\_array\_index]:  
 if matched\_non\_single\_element:  
 result += '\n' + union\_array\_index + str(index) + '\n\n' # start complex element  
 result += '|||\n'  
 result += '|---|---|\n'  
 for atr in element['attributes']:  
 result += '| \*' + atr + '\* | \*' + element['attributes'][atr] + '\* |\n'  
 if element['text']:  
 result += '| ' + element['name'] + ' | ' + element['text'] + ' |\n'  
 if element['children']:  
 result += to\_markdown\_str(element['children'])  
 index += 1  
 if matched\_non\_single\_element:  
 result += '\n'  
 result += '\n'  
 else:  
 for element in elements\_xml:  
 if len(element['children']) > 1:  
 result += '#' \* current\_head\_count + ' ' + element['name'] + ':\n\n' # single element  
 result += '|||\n'  
 result += '|---|---|\n'  
 for atr in element['attributes']:  
 result += '| \*' + atr + '\* | \*' + element['attributes'][atr] + '\* |\n' # attr  
 if element['text']:  
 result += '| ' + element['name'] + ' | ' + element['text'] + ' |\n' # text  
 if element['children']:  
 result += to\_markdown\_str(element['children'], current\_head\_count + 1)  
 if len(element['children']) > 1:  
 result += '\n'  
 result += '\n'  
 return result.rstrip()  
  
  
def union\_by\_name(list\_data):  
 result\_dict = {}  
  
 for value in list\_data:  
 name = value["name"]  
 if name in result\_dict:  
 result\_dict[name].append(value)  
 else:  
 result\_dict[name] = [value]  
  
 return result\_dict  
  
  
elements\_from\_xml = parse\_xml(open('input.xml').read())  
print(elements\_from\_xml)  
open('output\_add4.md', mode='w').write(to\_markdown\_str(elements\_from\_xml))

### Содержимое результирующего файла:

output\_add4.md:

# schedule:  
  
|||  
|---|---|  
| group | P3106 |  
| semester | Spring |  
## day:  
  
|||  
|---|---|  
| \*dayname\* | \*Среда\* |  
  
subject1  
  
|||  
|---|---|  
| \*type\* | \*Лабораторная\* |  
| time | 13:30-15:00 |  
| week | четная |  
| name | Программирование |  
| teacher | Байрамова Хумай Бахруз Кызы |  
| place | Ауд. 1338 (Бывш. 381), Кронверкский пр., д.49, лит.А |  
| weeks | 2, 4, 6, 8, 10, 12, 14, 16 |  
| mode | Очно - дистанционный |  
  
subject2  
  
|||  
|---|---|  
| \*type\* | \*Лабораторная\* |  
| time | 13:30-15:00 |  
| week | нечетная |  
| name | Основы профессиональной деятельности |  
| teacher | Афанасьев Дмитрий Борисович |  
| place | Ауд. 2435/5 (Бывш. 431Г), Кронверкский пр., д.49, лит.А |  
| weeks | 1, 3, 7, 11, 13, 15, 17, 18 |  
| mode | Очный |  
  
subject3  
  
|||  
|---|---|  
| \*type\* | \*Лабораторная\* |  
| time | 15:20-16:50 |  
| week | четная |  
| name | Программирование |  
| teacher | Байрамова Хумай Бахруз Кызы |  
| place | Ауд. 1338 (Бывш. 381), Кронверкский пр., д.49, лит.А |  
| weeks | 2, 4, 6, 8, 10, 12, 14, 16 |  
| mode | Очно - дистанционный |  
  
subject4  
  
|||  
|---|---|  
| \*type\* | \*Лабораторная\* |  
| time | 15:20-16:50 |  
| week | нечетная |  
| name | Основы профессиональной деятельности |  
| teacher | Афанасьев Дмитрий Борисович |  
| place | Ауд. 2435/5 (Бывш. 431Г), Кронверкский пр., д.49, лит.А |  
| weeks | 1, 3, 7, 11, 13, 15, 17, 18 |  
| mode | Очный |

### Микровывод:

Так как формат XML из себя представляет набор «ключ-значение», то лучшей интерпретацией данного формата мне показалась таблица. В Markdown таблицы делаются с помощью вертикальных (|) и минимум трех горизонтальных (-) линий. Все, что отделяют горизонтальные линии: заголовок и содержание таблицы. Дополнительно бывшие атрибуты тегов XML были помечены курсивным текстом.

# Вывод

Во время выполнения данной лабораторной работы я ознакомился с различными языками разметки, их применением в сегодняшнем мире, а также написал несколько скриптов, переводящие данные из одного языка разметки в другой.

# Список литературы

1. Балакшин П.В., Соснин В.В., Калинин И.В., Малышева Т.А., Раков С.В., Рущенко Н.Г., Дергачев А.М. Информатика: лабораторные работы и тесты: Учебно-методическое пособие / Рецензент: Поляков В.И. - Санкт-Петербург: Университет ИТМО, 2019. - 56 с. - экз. - Режим доступа: <https://books.ifmo.ru/book/2248/informatika:_laboratornye_raboty_i_testy:_uchebno-metodicheskoe_posobie_/_recenzent:_polyakov_v.i..htm>
2. Форма Бэкуса-Наура [Электронный ресурс]: Википедия. Режим доступа: <https://ru.wikipedia.org/wiki/Форма_Бэкуса_—_Наура> (дата обращения: 11/05/2023)
3. Markdown Extended Syntax [Электронный ресурс]. – Режим доступа: <https://www.markdownguide.org/extended-syntax/>

(дата обращения: 11/05/2023)