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**

по дисциплине

«ИНФОРМАТИКА»

Исследование протоколов, форматов обмена информацией и языков разметки документов

Вариант №07

***Выполнил:***Рязанов Никита Сергеевич

студент группы P3107

***Проверил:***Балакшин Павел Валерьевич

кандидат технических наук, доцент факультета ПИиКТ

**Содержание**

[Задание 3](#_Toc182026019)

[Основные этапы вычисления 5](#_Toc182026020)

[Заключение 22](#_Toc182026021)

[Список литературы 23](#_Toc182026022)

Задание

1. Определить номер варианта.
2. Изучить форму Бэкуса-Наура.
3. Изучить основные принципы организации формальных грамматик.
4. Изучить особенности языков разметки/форматов JSON, YAML, XML.
5. Понять устройство страницы с расписанием на примере расписания лектора.
6. Исходя из структуры расписания конкретного дня, сформировать файл с расписанием в формате, указанном в задании в качестве исходного.
7. 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. Нельзя использовать готовые библиотеки, в том числе регулярные выражения и библиотеки.
8. 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. Объяснение должно быть отражено в отчёте.
9. Дополнительное задание №2:
   1. Переписать исходный код, добавив в него использование регулярных выражений.
   2. Сравнить полученные результаты и объяснить их сходство/различие. Объяснение должно быть отражено в отчёте.
10. Дополнительное задание №3:
    1. 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.
    2. Проверку осуществить как минимум для расписания с двумя учебными днями по два занятия в каждом.
    3. Сравнить полученные результаты и объяснить их сходство/различие. Объяснение должно быть отражено в отчёте.
11. Дополнительное задание №4:
    1. Используя свою исходную программу из обязательного задания и программы из дополнительных заданий, сравнить стократное время выполнения парсинга и конвертации в цикле.
    2. Проанализировать полученные результаты и объяснить их сходство/различие. Объяснение должно быть отражено в отчёте.
12. Дополнительное задание №5:
    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. Объяснение должно быть отражено в отчёте.

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер варианта** | **Исходный формат** | **Результирующий формат** | **Дни недели** |
| 7 | XML | JSON | Вторник |
| Пятница |

Таблица 1. Форматы и дни недели в соответствии с вариантом

Основные этапы вычисления

Обязательное задание

Получив необходимые теоретические знания, было решено составить расписание конкретных для данного варианта дней в заданном формате. Получившийся файл *my\_schedule.xml* имеет следующий вид:

|  |
| --- |
| 1. <?xml version="1.0" encoding="UTF-8"?> 2. <schedule> 3. <Tuesday> 4. <lesson1> 5. <name> 6. Математический анализ 7. </name> 8. <type>Практика</type> 9. <group>Мат Ан 2.2</group> 10. <teacher>Милюшин Александр Сергеевич</teacher> 11. <time> 12. <start>13:30</start> 13. <end>15:00</end> 14. </time> 15. <weeks>1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16</weeks> 16. <place> 17. <building>ул.Ломоносова, д.9, лит. М</building> 18. <room>1314</room> 19. </place> 20. <format>Очно</format> 21. </lesson1> 22. </Tuesday> 24. <Friday> 25. <lesson1> 26. <name>Линейная алгебра</name> 27. <type>Практика</type> 28. <group>Лин Алг 3.3</group> 29. <teacher>Ефремов Илья Дмитриевич</teacher> 30. <time> 31. <start>10:00</start> 32. <end>11:30</end> 33. </time> 34. <weeks>1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16</weeks> 35. <place> 36. <building>Кронверкский пр., д.49, лит.А</building> 37. <room>2201</room> 38. </place> 39. <format>Очно</format> 40. </lesson1> 42. <lesson2> 43. <name>Английский язык</name> 44. <type>Практика</type> 45. <group>АЯ-B1.2/49</group> 46. <teacher>Бревнова Татьяна Сергеевна</teacher> 47. <time> 48. <start>15:20</start> 49. <end>16:50</end> 50. </time> 51. <weeks>1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16</weeks> 52. <place> 53. <building>ул.Ломоносова, д.9, лит. М</building> 54. <room>3308</room> 55. </place> 56. <format>Очно</format> 57. </lesson2> 59. <lesson3> 60. <name>Английский язык</name> 61. <type>Практика</type> 62. <group>АЯ-B1.2/49</group> 63. <teacher>Бревнова Татьяна Сергеевна</teacher> 64. <time> 65. <start>17:00</start> 66. <end>18:30</end> 67. </time> 68. <weeks>1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16</weeks> 69. <place> 70. <building>ул.Ломоносова, д.9, лит. М</building> 71. <room>3308</room> 72. </place> 73. <format>Очно</format> 74. </lesson3> 75. </Friday> 76. </schedule> |

Код 1. Расписание в формате XML

Далее была написана программа на языке программирования Python, которая осуществляет конвертацию файла из исходного формата (XML) в новый (JSON) путем замены метасимволов без использования регулярных выражений и различных библиотек. Листинг получившейся программы представлен ниже:

|  |
| --- |
| 1. JSON\_INDENT = ' ' \* 4 2. def xml\_list\_to\_string(xml\_list): 3. """ 4. Converts XML (represented as a list of strings) to string with removed leading and trailing whitespaces. 5. """ 6. return ''.join([l.strip() if l.rstrip().endswith('>') else l.lstrip().rstrip('\n') 7. for l in xml\_list 8. if (l.strip() and not(l.strip().endswith('?>'))) 9. ]) 10. def convert\_raw(xml\_list): 11. """ 12. Converts XML (represented as a list of strings) to JSON (represented as a string). 13. This method doesn't use any libraries, regex and etc. 14. """ 15. # Convert XML to string 16. xml\_str = xml\_list\_to\_string(xml\_list) 18. # List of JSON strings 19. json\_list = ['{'] 20. # Index of current bracket 21. br = -1 22. # Counter of unclosed tags 23. unclosed\_tags = 0 24. # List of tag brackets in order 25. brs = [] 26. for i in range(len(xml\_str)): 27. if xml\_str[i:i+2] == '</': 28. brs.append('</') 29. elif xml\_str[i] == '<': 30. brs.append('<') 32. i = 0 33. while i < len(xml\_str): 34. if xml\_str[i:i + 2] == '</': 35. br += 1 36. unclosed\_tags -= 1 37. # If last bracket then exit loop 38. if (br + 1) >= len(brs): 39. break 40. # If next tag is closing tag too then this tag is nested, so close parent tag with curly bracket 41. if brs[br + 1] == '</': 42. json\_list.append(JSON\_INDENT \* unclosed\_tags + '}' + \ 43. # If there is another tag after this then add comma 44. (',' if (br + 1 < len(brs) and brs[br + 1] == '<') else '')) 45. # If next tag is opening tag then add comma 46. else: 47. json\_list[-1] += ',' 48. # Move to end of this tag 49. i = xml\_str.index('>', i + 1) 50. elif xml\_str[i] == '<': 51. br += 1 52. unclosed\_tags += 1 53. # Add formatted tag to key 54. json\_list.append(JSON\_INDENT \* unclosed\_tags + '"' + xml\_str[i + 1: xml\_str.index('>', i + 1)] + '": ') 55. # Move to end of this tag 56. i = xml\_str.index('>', i + 1) 57. elif xml\_str[i] == '>': 58. # If next tag is opening tag too then add curly bracket (because next tag is nested in this) 59. if brs[br] == '<' and brs[br + 1] == '<': 60. json\_list[-1] += '{' 61. # Increment index 62. i += 1 63. else: 64. # Add value of current tag to last key 65. json\_list[-1] += '"' + xml\_str[i:xml\_str.index('</', i + 1)] + '"' 66. # Move to closing tag 67. i = xml\_str.index('</', i + 1) 69. # Add closing bracket of JSON file 70. json\_list.append('}') 71. return '\n'.join(json\_list) 72. if \_\_name\_\_ == '\_\_main\_\_': 73. xml\_list = [] 74. with open(r'data\my\_schedule.xml', 'r', encoding='utf-8') as f: 75. xml\_list = f.readlines() 76. with open(r'output\my\_schedule\_raw.json', 'w', encoding='utf-8') as f: 77. f.writelines( 78. convert\_raw(xml\_list) 79. ) |

Код 2. Программа для конвертации с помощью замены метасимволов

Результатом выполнения данной программы является конвертированный из формата XML файл расписания в формате JSON. Полученный файл *my\_schedule\_raw.json* выглядит следующим образом:

|  |
| --- |
| 1. { 2. "schedule": { 3. "Tuesday": { 4. "lesson1": { 5. "name": "Математический анализ", 6. "type": "Практика", 7. "group": "Мат Ан 2.2", 8. "teacher": "Милюшин Александр Сергеевич", 9. "time": { 10. "start": "13:30", 11. "end": "15:00" 12. }, 13. "weeks": "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16", 14. "place": { 15. "building": "ул.Ломоносова, д.9, лит. М", 16. "room": "1314" 17. }, 18. "format": "Очно" 19. } 20. }, 21. "Friday": { 22. "lesson1": { 23. "name": "Линейная алгебра", 24. "type": "Практика", 25. "group": "Лин Алг 3.3", 26. "teacher": "Ефремов Илья Дмитриевич", 27. "time": { 28. "start": "10:00", 29. "end": "11:30" 30. }, 31. "weeks": "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16", 32. "place": { 33. "building": "Кронверкский пр., д.49, лит.А", 34. "room": "2201" 35. }, 36. "format": "Очно" 37. }, 38. "lesson2": { 39. "name": "Английский язык", 40. "type": "Практика", 41. "group": "АЯ-B1.2/49", 42. "teacher": "Бревнова Татьяна Сергеевна", 43. "time": { 44. "start": "15:20", 45. "end": "16:50" 46. }, 47. "weeks": "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16", 48. "place": { 49. "building": "ул.Ломоносова, д.9, лит. М", 50. "room": "3308" 51. }, 52. "format": "Очно" 53. }, 54. "lesson3": { 55. "name": "Английский язык", 56. "type": "Практика", 57. "group": "АЯ-B1.2/49", 58. "teacher": "Бревнова Татьяна Сергеевна", 59. "time": { 60. "start": "17:00", 61. "end": "18:30" 62. }, 63. "weeks": "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16", 64. "place": { 65. "building": "ул.Ломоносова, д.9, лит. М", 66. "room": "3308" 67. }, 68. "format": "Очно" 69. } 70. } 71. } 72. } |

Код 3. Расписание в формате JSON в результате convert\_raw

Дополнительное задание №1

Для выполнения данного дополнительного задания была найдена библиотека под названием *xmltodict*[[1]](#footnote-1),которая переводит XML файл в упорядоченный словарь. Дальше этот словарь легко перевести в формат JSON с помощью одноименной библиотеки, встроенной в язык программирования Python. Листинг программы с использованием вышеупомянутых библиотек представлен ниже:

|  |
| --- |
| 1. import xmltodict 2. import json 3. JSON\_INDENT = ' ' \* 4 4. def convert\_lib(xml\_list): 5. """ 6. Converts XML (represented as a list of strings) to JSON (represented as a string). 7. This method uses standard json library and xmltodict package by martinblech. 8. """ 9. return json.dumps( 10. xmltodict.parse(''.join(xml\_list)), 11. ensure\_ascii=False, 12. indent=JSON\_INDENT 13. ) 14. if \_\_name\_\_ == '\_\_main\_\_': 15. xml\_list = [] 16. with open(r'data\my\_schedule.xml', 'r', encoding='utf-8') as f: 17. xml\_list = f.readlines() 18. with open(r'output\my\_schedule\_lib.json', 'w', encoding='utf-8') as f: 19. f.writelines( 20. convert\_lib(xml\_list) 21. ) |

Код 4. Программа для конвертации с использованием библиотек

Результатом выполнения данной программы является файл расписания в формате JSON. Полученный файл *my\_schedule\_lib.json* выглядит следующим образом:

|  |
| --- |
| 1. { 2. "schedule": { 3. "Tuesday": { 4. "lesson1": { 5. "name": "Математический анализ", 6. "type": "Практика", 7. "group": "Мат Ан 2.2", 8. "teacher": "Милюшин Александр Сергеевич", 9. "time": { 10. "start": "13:30", 11. "end": "15:00" 12. }, 13. "weeks": "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16", 14. "place": { 15. "building": "ул.Ломоносова, д.9, лит. М", 16. "room": "1314" 17. }, 18. "format": "Очно" 19. } 20. }, 21. "Friday": { 22. "lesson1": { 23. "name": "Линейная алгебра", 24. "type": "Практика", 25. "group": "Лин Алг 3.3", 26. "teacher": "Ефремов Илья Дмитриевич", 27. "time": { 28. "start": "10:00", 29. "end": "11:30" 30. }, 31. "weeks": "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16", 32. "place": { 33. "building": "Кронверкский пр., д.49, лит.А", 34. "room": "2201" 35. }, 36. "format": "Очно" 37. }, 38. "lesson2": { 39. "name": "Английский язык", 40. "type": "Практика", 41. "group": "АЯ-B1.2/49", 42. "teacher": "Бревнова Татьяна Сергеевна", 43. "time": { 44. "start": "15:20", 45. "end": "16:50" 46. }, 47. "weeks": "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16", 48. "place": { 49. "building": "ул.Ломоносова, д.9, лит. М", 50. "room": "3308" 51. }, 52. "format": "Очно" 53. }, 54. "lesson3": { 55. "name": "Английский язык", 56. "type": "Практика", 57. "group": "АЯ-B1.2/49", 58. "teacher": "Бревнова Татьяна Сергеевна", 59. "time": { 60. "start": "17:00", 61. "end": "18:30" 62. }, 63. "weeks": "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16", 64. "place": { 65. "building": "ул.Ломоносова, д.9, лит. М", 66. "room": "3308" 67. }, 68. "format": "Очно" 69. } 70. } 71. } 72. } |

Код 5. Расписание в формате JSON в результате convert\_lib

Если сравнивать полученные файлы JSON двух разных методов конвертации, то они идентичны. Связано это с тем, что первый метод был специально написан под исходный XML файл, а второй метод использует готовую библиотеку, которая должна работать гарантированно для любого XML файла, и так как файл с расписанием является довольно простым по структуре, то разница в полученных результатах отсутствует.

Дополнительное задание №2

Для выполнения данного дополнительного задания был переписан первоначальный метод конвертации, но теперь с использованием регулярных выражений. Было решено взять в качестве основного паттерна следующее: **<(?P<tag>.\*?)>(.\*?)</(?P=tag)>**. Логика данного решения заключается в том, что мы берем тег со всем, что находится внутри него, затем рекурсивно смотрим, есть внутри тега другие теги по этому же паттерну. Во время рекурсии все найденные теги также преобразовываются в соответствии с форматом JSON. Листинг программы с использованием регулярных выражений представлен ниже:

|  |
| --- |
| 1. import re 2. JSON\_INDENT = ' ' \* 4 3. TAG\_PATTERN = re.compile(r'<(?P<tag>.\*?)>(.\*?)</(?P=tag)>') 4. def xml\_list\_to\_string(xml\_list): 5. """ 6. Converts XML (represented as a list of strings) to string with removed leading and trailing whitespaces. 7. """ 8. return ''.join([l.strip() if l.rstrip().endswith('>') else l.lstrip().rstrip('\n') 9. for l in xml\_list 10. if (l.strip() and not(l.strip().endswith('?>'))) 11. ]) 12. def \_\_replace\_tag(match, unclosed\_tags = 1): 13. """ 14. Recursively replaces XML tag to JSON format (represented as a string). 15. """ 16. # Check that match is not None 17. if match: 18. # Find all tags inside this tag 19. next\_matches = TAG\_PATTERN.findall(match[1]) 21. # If there are nested tags 22. if next\_matches: 23. scope = [] 24. # Add curly bracket as we have nested tags 25. scope.append(f'{JSON\_INDENT \* unclosed\_tags}"{match[0]}":' + ' {') 26. # Recursively add nested tags 27. for next\_match in next\_matches: 28. scope.append(f'{\_\_replace\_tag(next\_match, unclosed\_tags + 1)},') 29. # Remove comma from last tag 30. scope[-1] = scope[-1][:-1] 31. # Close current tag 32. scope.append(f'{JSON\_INDENT \* unclosed\_tags}' + '}') 33. # Return current scope as a string 34. return '\n'.join(scope) 36. # If there are no nested tags return formatted tag as a key-value pair 37. return f'{JSON\_INDENT \* unclosed\_tags}"{match[0]}": "{match[1]}"' 38. def convert\_regex(xml\_list): 39. """ 40. Converts XML (represented in list of strings) to JSON (represented in string). 41. This method uses regex. 42. """ 43. # Convert XML to string 44. xml\_str = xml\_list\_to\_string(xml\_list) 46. json\_str = '{\n' 48. # Find root tag 49. main\_tag = TAG\_PATTERN.fullmatch(xml\_str) 50. main\_tag = (main\_tag.group(1), main\_tag.group(2)) 51. # Format root tag 52. json\_str += \_\_replace\_tag(main\_tag) 53. # (Add closing bracket and) return JSON as a string 54. return json\_str + '\n}' 56. if \_\_name\_\_ == '\_\_main\_\_': 57. xml\_list = [] 58. with open(r'data\my\_schedule.xml', 'r', encoding='utf-8') as f: 59. xml\_list = f.readlines() 60. with open(r'output\my\_schedule\_regex.json', 'w', encoding='utf-8') as f: 61. f.writelines( 62. convert\_regex(xml\_list) 63. ) |

Код 6. Программа для конвертации с использованием регулярных выражений

Результатом выполнения данной программы является файл расписания в формате JSON. Полученный файл *my\_schedule\_regex.json* выглядит следующим образом:

|  |
| --- |
| 1. { 2. "schedule": { 3. "Tuesday": { 4. "lesson1": { 5. "name": "Математический анализ", 6. "type": "Практика", 7. "group": "Мат Ан 2.2", 8. "teacher": "Милюшин Александр Сергеевич", 9. "time": { 10. "start": "13:30", 11. "end": "15:00" 12. }, 13. "weeks": "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16", 14. "place": { 15. "building": "ул.Ломоносова, д.9, лит. М", 16. "room": "1314" 17. }, 18. "format": "Очно" 19. } 20. }, 21. "Friday": { 22. "lesson1": { 23. "name": "Линейная алгебра", 24. "type": "Практика", 25. "group": "Лин Алг 3.3", 26. "teacher": "Ефремов Илья Дмитриевич", 27. "time": { 28. "start": "10:00", 29. "end": "11:30" 30. }, 31. "weeks": "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16", 32. "place": { 33. "building": "Кронверкский пр., д.49, лит.А", 34. "room": "2201" 35. }, 36. "format": "Очно" 37. }, 38. "lesson2": { 39. "name": "Английский язык", 40. "type": "Практика", 41. "group": "АЯ-B1.2/49", 42. "teacher": "Бревнова Татьяна Сергеевна", 43. "time": { 44. "start": "15:20", 45. "end": "16:50" 46. }, 47. "weeks": "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16", 48. "place": { 49. "building": "ул.Ломоносова, д.9, лит. М", 50. "room": "3308" 51. }, 52. "format": "Очно" 53. }, 54. "lesson3": { 55. "name": "Английский язык", 56. "type": "Практика", 57. "group": "АЯ-B1.2/49", 58. "teacher": "Бревнова Татьяна Сергеевна", 59. "time": { 60. "start": "17:00", 61. "end": "18:30" 62. }, 63. "weeks": "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16", 64. "place": { 65. "building": "ул.Ломоносова, д.9, лит. М", 66. "room": "3308" 67. }, 68. "format": "Очно" 69. } 70. } 71. } 72. } |

Код 7. Расписание в формате JSON в результате convert\_regex

Если сравнивать полученные файлы JSON исходного метода конвертации с методом, использующий регулярные выражения, то они идентичны. Связано это с тем, что оба метода были специально написаны под XML файл с расписанием, тем самым разница в полученных результатах отсутствует.

Дополнительное задание №3

Для выполнения данного дополнительного задания требуется использование формальных грамматик. Опишем с помощью них элементы XML файлов, которые с помощью лексера (токенайзера) будут переведены в некие токены:

|  |
| --- |
| 1. import re 2. # Formal Grammar 3. CHAR = r'.' 4. WHITESPACE = r'\s' 5. NAME = r'[a-zA-Z][a-zA-Z0-9\_]\*' 6. COMMENT = rf'<!--({CHAR}\*?)-->' 7. EQ = rf'{WHITESPACE}?={WHITESPACE}?' 8. ATTRIBUTE\_VALUE = rf'"[^<\&"]\*?"' 9. ATTRIBUTE = rf'({NAME}){EQ}({ATTRIBUTE\_VALUE})' 10. OPEN\_TAG = rf'<({NAME})(({WHITESPACE}{ATTRIBUTE})\*?){WHITESPACE}?>' 11. CLOSE\_TAG = rf'</({NAME}){WHITESPACE}?>' 12. CONTENT = rf'([^<\&]\*)?' 13. SELF\_CLOSING\_TAG = rf'<({NAME})(({WHITESPACE}{ATTRIBUTE})\*?){WHITESPACE}?/>' 14. TOKENS = [ 15. # (Regex, Token type) 16. (re.compile(COMMENT), 'COMMENT'), 17. (re.compile(OPEN\_TAG), 'OPEN\_TAG'), 18. (re.compile(CLOSE\_TAG), 'CLOSE\_TAG'), 19. (re.compile(SELF\_CLOSING\_TAG), 'SELF\_CLOSING\_TAG'), 20. (re.compile(CONTENT), 'CONTENT') 21. ] 22. class Tokenizer: 23. # String to tokenize 24. \_string: str 25. # Index of current position 26. \_ind: int 28. def \_\_init\_\_(self, string): 29. self.\_string = string 30. self.\_ind = 0 32. def has\_tokens\_left(self): 33. """ 34. Checks if there are tokens left in the string. 35. """ 36. return self.\_ind < len(self.\_string) 38. def format\_token(self, token\_type, match): 39. """ 40. Formats token into dictionary. 41. """ 42. # Create dictionary with token type 43. token = { 44. 'type': token\_type 45. } 46. if token\_type == 'OPEN\_TAG' or token\_type == 'SELF\_CLOSING\_TAG': 47. # Add name of tag to dictionary 48. token['name'] = match.group(1) 49. # Add attributes to dictionary 50. if (token\_type == 'OPEN\_TAG' or token\_type == 'SELF\_CLOSING\_TAG') and match.group(2): 51. token['attributes'] = {att[0]:(att[1][1:-1] if att[1][0] == '"' and att[1][-1] == '"' else att[1]) 52. for att in re.findall(ATTRIBUTE, match.group(2))} 53. elif token\_type == 'CLOSE\_TAG': 54. # Add name of tag to dictionary 55. token['name'] = match.group(1) 56. elif token\_type == 'CONTENT': 57. # Add value to dictionary 58. token['value'] = match.group(0) 59. elif token\_type == 'COMMENT': 60. # Add value to dictionary 61. token['value'] = match.group(1) 62. return token 63. def get\_next\_token(self): 64. """ 65. Finds next token in the string. 66. Returns None if token is not found. 67. """ 68. # Get string from current position 69. string = self.\_string[self.\_ind:] 70. # Try to match string with one of the tokens 71. for regex, token\_type in TOKENS: 72. if match := regex.match(string): 73. # Move current position 74. self.\_ind += len(match.group(0)) 75. # Return formatted token 76. return self.format\_token(token\_type, match) 77. # If token is not found 78. return None 80. def tokenize(self): 81. # List of tokens in string 82. tokens = [] 83. # While there are tokens left 84. while self.has\_tokens\_left(): 85. if token := self.get\_next\_token(): 86. # Add token to list 87. tokens.append(token) 88. else: 89. # If token is not found 90. raise Exception(f'Unrecognized token from index {self.\_ind}: {self.\_string[self.\_ind:self.\_ind+10]}...') 91. # Add end of file token 92. tokens.append({'type': 'EOF'}) 93. return tokens |

Код 8. Лексер

Затем будем парсить токены в упорядоченные словари, получив на выходе некое подобие Abstract Syntax Tree (AST):

|  |
| --- |
| 1. from tokenizer import Tokenizer 2. from collections import OrderedDict 3. class Parser(): 4. # List of tokens of given string 5. \_tokens: list 6. # Dictionary of parsed tokens 7. \_dict: OrderedDict 8. # Path to current scope 9. \_path: list 10. def \_\_init\_\_(self, string): 11. tokenizer = Tokenizer(string) 12. self.\_tokens = tokenizer.tokenize() 13. self.\_dict = OrderedDict() 14. self.\_path = [] 15. def \_\_parse\_token(self, token\_ind = 0): 16. # Get current token 17. token = self.\_tokens[token\_ind] 18. # Get previous and current scopes 19. prev\_scope = self.\_dict 20. cur\_scope = self.\_dict 21. if self.\_path: 22. for p in self.\_path[:-1]: 23. prev\_scope = prev\_scope[p] 24. cur\_scope = cur\_scope[p] 25. cur\_scope = cur\_scope[self.\_path[-1]] 26. if token['type'] == 'OPEN\_TAG': 27. # Add path to current scope 28. self.\_path.append(token['name']) 29. # Check if scope already exists 30. if token['name'] not in cur\_scope: 31. # If not, check if it is array of values 32. if isinstance(cur\_scope, list): 33. # If it is array, add new element to the end 34. cur\_scope.append(OrderedDict()) 35. # Add path to the element 36. self.\_path.append(-1) 37. else: 38. # If it is not array, create new scope 39. cur\_scope[token['name']] = OrderedDict() 40. else: 41. # If scope exists, check if it is already array of scopes 42. if isinstance(cur\_scope[token['name']], list): 43. # If it is array, add new scope to the end 44. cur\_scope[token['name']].append(OrderedDict()) 45. else: 46. # If it is not array, format to array of scopes 47. cur\_scope[token['name']] = [cur\_scope[token['name']], OrderedDict()] 48. # Add path to the new scope (last element of the array of scopes) 49. self.\_path.append(-1) 51. # If there are attributes, add them to the scope 52. if 'attributes' in token: 53. cur\_scope[token['name']].update({f'@{k}':v for k, v in token['attributes'].items()}) 54. # Move to the next token 55. return self.\_\_parse\_token(token\_ind + 1) 56. elif token['type'] == 'CLOSE\_TAG': 57. # If current scope has no any value, set it to None 58. if len(prev\_scope[self.\_path[-1]]) == 0: 59. prev\_scope[self.\_path[-1]] = None 60. # If current scope is element of an array, additionally delete path to its array 61. # Else, delete path to the current scope 62. if self.\_path[-1] == -1: 63. self.\_path.pop() 64. self.\_path.pop() 65. # Move to the next token 66. return self.\_\_parse\_token(token\_ind + 1) 67. elif token['type'] == 'CONTENT': 68. # If current scope is an array, add value to the end 69. # Else, set the value 70. if isinstance(prev\_scope[self.\_path[-1]], list): 71. prev\_scope[self.\_path[-1]].append(token['value']) 72. else: 73. prev\_scope[self.\_path[-1]] = token['value'] 74. # Move to the next token 75. return self.\_\_parse\_token(token\_ind + 1) 76. elif token['type'] == 'COMMENT': 77. # Skip comment and move to the next token 78. return self.\_\_parse\_token(token\_ind + 1) 79. elif token['type'] == 'SELF\_CLOSING\_TAG': 80. # Set scope of the current token to None as it is self-closing 81. cur\_scope[token['name']] = None 82. # If there are attributes, add them to the scope 83. if 'attributes' in token: 84. cur\_scope[token['name']] = OrderedDict({f'@{k}':v for k, v in token['attributes'].items()}) 85. # Move to the next token 86. return self.\_\_parse\_token(token\_ind + 1) 87. elif token['type'] == 'EOF': 88. # Return the dictionary as we parsed all tokens 89. return self.\_dict 90. def parse(self): 91. """ 92. Returns parsed string in the form of dictionary. 93. """ 94. return self.\_\_parse\_token() |

Код 9. Парсер

Полученный упорядоченный словарь можно легко преобразовать в формат JSON при помощи стандартной библиотеки:

|  |
| --- |
| 1. import json 2. from parser import Parser 3. JSON\_INDENT = ' ' \* 4 4. def xml\_list\_to\_string(xml\_list): 5. """ 6. Converts XML (represented as a list of strings) to string with removed leading and trailing whitespaces. 7. """ 8. return ''.join([l.strip() if l.rstrip().endswith('>') else l.lstrip().rstrip('\n') 9. for l in xml\_list 10. if (l.strip() and not(l.strip().endswith('?>'))) 11. ]) 12. def convert\_formal\_grammar(xml\_list): 13. """ 14. Converts XML (represented in list of strings) to JSON (represented in string). 15. This method uses formal grammar. 16. """ 17. # Convert XML to string 18. xml\_str = xml\_list\_to\_string(xml\_list) 20. parser = Parser(xml\_str) 22. return json.dumps( 23. parser.parse(), 24. ensure\_ascii=False, 25. indent=JSON\_INDENT 26. ) |

Код 10. Программа для конвертации с использованием формальных грамматик

Для сравнения возможностей каждого метода были написаны различные тесты, которые покрывают определенные элементы, встречаемые в формате XML:

* 1. Расписание
  2. Базовый тег
  3. Массивы
  4. Атрибуты
  5. Комментарии
  6. Пустые теги
  7. Совокупность всех этих элементов

Полный листинг программы для тестирования доступен в репозитории:

* <https://github.com/s4dnex/itmo-labs/tree/main/informatics/lab4>

Результаты тестирования методов конвертации представлены ниже (см. Таблица 2)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Расписание** | **Базовый тег** | **Массивы** | **Атрибуты** | **Комментарии** | **Пустые теги** | **Все в одном** |
| **Raw** | PASS | PASS | FAIL | FAIL | FAIL | FAIL | FAIL |
| **Library** | PASS | PASS | PASS | PASS | PASS | PASS | PASS |
| **RegEx** | PASS | PASS | FAIL | FAIL | FAIL | FAIL | FAIL |
| **Formal Grammar** | PASS | PASS | PASS | PASS | PASS | PASS | PASS |

Таблица 2. Результаты тестов

Из таблицы видно, что методы с заменой метасимволов и с использованием регулярных выражений не покрывают многие тесты, в то время как методы, использующие готовые библиотеки или формальную грамматику, успешно их проходят. Причиной этого является то, что методы, которые были написаны чисто для перевода расписания, опирались только на элементы, которые присутствовали в нем, то есть в учет не брались другие элементы, которые присутствуют в тестах.

Дополнительное задание №4

Для сравнения времени выполнения методов была написана простая программа для измерения стократного выполнения конвертации заданным методом с использованием стандартной библиотеки *time*:

|  |
| --- |
| 1. import time 2. STRING\_FORMAT = '{0:<20} {1:<20}' 3. def test\_time(func): 4. with open(r'data\my\_schedule.xml', 'r', encoding='utf-8') as f: 5. xml\_list = f.readlines() 6. start = time.perf\_counter() 7. for i in range(100): 8. func(xml\_list) 9. end = time.perf\_counter() 10. return end - start 11. if \_\_name\_\_ == '\_\_main\_\_': 12. print(STRING\_FORMAT.format('Method', 'Time (in seconds)')) 13. print(STRING\_FORMAT.format('Raw', test\_time(convert\_raw))) 14. print(STRING\_FORMAT.format('Library', test\_time(convert\_lib))) 15. print(STRING\_FORMAT.format('RegEx', test\_time(convert\_regex))) 16. print(STRING\_FORMAT.format('Formal Grammar', test\_time(convert\_formal\_grammar))) |

Код 11. Программа для измерения времени выполнения методов

Результаты измерения времени выполнения методов конвертации представлены ниже (см. Рисунок 1):

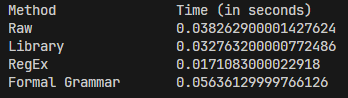


Рисунок 1. Результаты измерения времени

Из результатов видно, что метод с использованием регулярных выражений оказался самым быстрым. Объяснить это можно тем, что заранее компилируется паттерн для дальнейшего поиска тегов, и тем, что данный метод был написан с упором на файл с расписанием, поэтому, как уже было подмечено, он не учитывает элементы, которые могут встретиться в других случаях и не тратит на их поиск дополнительное время.

Примечательно, что следующий лучший результат по времени показывает метод, который использует сторонние библиотеки. Предполагалось, что из-за общего назначения данный метод будет тратить больше всего времени, но это оказалось не так. Можно предположить, что данная библиотека либо очень хорошо оптимизирована, либо, возможно, использует какие-то нативные решения на языке программирования C, что существенно ускоряет ее работу.

Один из худших результатов по времени имеет метод с заменой метасимволов. Связано это может быть с тем, что данный метод имеет очень много условных выражений, несколько циклов и т. д., что явно негативно влияет на скорость работы метода.

Ожидаемо, что самый худший результат показал метод с использованием формальных грамматик, ведь данный метод сначала преобразует XML в токены, затем токены в подобие AST и только после этого в JSON.

Дополнительное задание №5

В качестве формата для конвертации был выбран TSV (Tab Separated Values) – текстовый формат для представления таблиц баз данных, где каждое поле записи отделяется от других с помощью символа табуляции. За основу был взят метод конвертации с использованием регулярных выражений:

|  |
| --- |
| 1. TAG\_PATTERN = re.compile(r'<(?P<tag>.\*?)>(.\*?)</(?P=tag)>') 2. def xml\_list\_to\_string(xml\_list): 3. """ 4. Converts XML (represented as a list of strings) to string with removed leading and trailing whitespaces. 5. """ 6. return ''.join([l.strip() if l.rstrip().endswith('>') else l.lstrip().rstrip('\n') 7. for l in xml\_list 8. if (l.strip() and not(l.strip().endswith('?>'))) 9. ]) 10. def \_\_generate\_tsv(match, path = ''): 11. """ 12. Recursively replaces XML tag to TSV format. 13. """ 14. global header, values 15. # Check that match is not None 16. if match: 17. # Find all tags inside this tag 18. next\_matches = TAG\_PATTERN.findall(match[1]) 20. # If there are nested tags 21. if next\_matches: 22. # Recursively add nested tags 23. for next\_match in next\_matches: 24. \_\_generate\_tsv(next\_match, path + match[0] + '.') 25. # If there are no nested tags return formatted tag as a TSV value 26. else: 27. header.append(path + match[0]) 28. values.append(match[1]) 29. def convert(xml\_list): 30. """ 31. Converts XML (represented in list of strings) to TSV (represented in string). 32. """ 33. global header, values 34. header = [] 35. values = [] 36. # Convert XML to string 37. xml\_str = xml\_list\_to\_string(xml\_list) 38. # Find root tag 39. main\_tag = TAG\_PATTERN.fullmatch(xml\_str) 40. main\_tag = (main\_tag.group(1), main\_tag.group(2)) 41. # Format root tag 42. \_\_generate\_tsv(main\_tag) 43. return '\t'.join(header) + '\n' + '\t'.join(values) 44. if \_\_name\_\_ == '\_\_main\_\_': 45. xml\_list = [] 46. with open(r'data\my\_schedule.xml', 'r', encoding='utf-8') as f: 47. xml\_list = f.readlines() 49. with open(r'output\my\_schedule.tsv', 'w', encoding='utf-8') as f: 50. f.writelines( 51. xml\_to\_tsv.convert(xml\_list) 52. ) |

Код 12. Программа для конвертации в формат TSV

С полным результатом конвертации в формат TSV можно ознакомиться в репозитории:

* <https://github.com/s4dnex/itmo-labs/tree/main/informatics/lab4>

Из-за особенности выбранного формата, каждый тег в формате XML имеет свою отдельную колонку, таким образом получается, что все значения хранятся в одной длинной строке.

Заключение

В ходе проделанной лабораторной работе были изучены новые аспекты, возможности и особенности языка программирования Python, было получено много нового интересного теоретического материала про формальные грамматики, было проведено ознакомление с такими языками разметки как XML и JSON и были написаны различные методы по парсингу и конвертации файлов.

Список литературы

1. Python Docs. Документация по языку программирования Python. Версия 3.10. [Электронный ресурс] – Режим доступа: <https://docs.python.org/3.10/>
2. Балакшин П.В., Соснин В.В. Информатика. Методическое пособие. – Режим доступа: [https://picloud.pw/media/resources/posts/2018/02/19/%D0%9C%D0%B5%D1%82%  
   D0%BE%D0%B4%D0%B8%D1%87%D0%BA%D0%B0.pdf](https://picloud.pw/media/resources/posts/2018/02/19/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D1%87%D0%BA%D0%B0.pdf)
3. W3C. Документация по синтаксису XML. [Электронный ресурс] – Режим доступа: <https://www.w3.org/TR/xml/#NT-Reference>

1. PyPI. Репозиторий с библиотеками для Python. Пакет xmltodict. [Электронный ресурс] – Режим доступа: <https://pypi.org/project/xmltodict/> [↑](#footnote-ref-1)