Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет программной инженерии и компьютерных технологий

Информатика

Лабораторная работа №4

Исследование протоколов, форматов обмена информацией и языков разметки документов

Вариант 2

Выполнила студент

Умарова Амина

Группа №Р3118

Преподаватель: Малышева Татьяна

Алексеевна

г. Санкт-Петербург

2024

Оглавление

[Задание 3](#_Toc187738190)

[Основные этапы решения 4](#_Toc187738191)

[Исходный файл JSON 4](#_Toc187738192)

[Решение основного задания 4](#_Toc187738193)

[Исходный код 4](#_Toc187738194)

[Результат программы 4](#_Toc187738195)

[Решение доп. задания 1 4](#_Toc187738196)

[Исходный код 4](#_Toc187738197)

[Сравнение результатов 4](#_Toc187738198)

[Решение доп. задания 2 4](#_Toc187738199)

[Решение доп. задания 3 4](#_Toc187738200)

[Решение доп. задания 4 4](#_Toc187738201)

[Решение доп. задания 5 4](#_Toc187738202)

[Заключение: 4](#_Toc187738203)

[Список использованных источников: 4](#_Toc187738204)

# 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).
   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.
5. Дополнительное задание №2 (позволяет набрать +10 процентов от максимального числа баллов БаРС за данную лабораторную).
   1. Переписать исходный код, добавив в него использование регулярных выражений.
   2. Сравнить полученные результаты и объяснить их сходство/различие. Объяснение должно быть отражено в отчёт
6. Дополнительное задание №3 (позволяет набрать +25 процентов от максимального числа баллов БаРС за данную лабораторную).
   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.
7. Дополнительное задание №4 (позволяет набрать +5 процентов от максимального числа баллов БаРС за данную лабораторную).
   1. Используя свою исходную программу из обязательного задания и программы из дополнительных заданий, сравнить стократное время выполнения парсинга + конвертации в цикле.
   2. 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).
   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
   2. Проанализировать полученные результаты, объяснить особенности использования формата. Объяснение должно быть отражено в отчёте.
9. Проверить, что все пункты задания выполнены и выполнены верно.
10. Написать отчёт о проделанной работе.
11. Подготовиться к устным вопросам на защите

# Основные этапы решения

*Вариант 2: JSON -> YAML*

## Исходный файл JSON

{

"even\_week": **true**,

"group\_name": "P3118",

"date": "Tuesday, 3 Dec 2024",

"schedule": [

{

"subject": "English",

"class\_format": "face-to-face",

"type": "practice",

"campus": "ул. Ломоносова 9",

"auditory": "3211",

"teacher": "Ермакова Елена Владиславовна",

"time": {

"period": **1**,

"from\_time": "08:20",

"to\_time": "09:50"

}

},

{

"subject": "English",

"class\_format": "face-to-face",

"type": "practice",

"campus": "ул. Ломоносова 9",

"auditory": "3211",

"teacher": "Ермакова Елена Владиславовна",

"time": {

"period": **2**,

"from\_time": "10:00",

"to\_time": "11:30"

}

},

{

"subject": "Discrete\_Math",

"class\_format": "face-to-face",

"type": "lecture",

"campus": "Кронверкский пр., д.49, лит.А",

"auditory": "1404",

"teacher": "Карпов Дмитрий Валерьевич",

"time": {

"period": **7**,

"from\_time": "18:40",

"to\_time": "20:10"

}

},

{

"subject": "Discrete\_Math",

"class\_format": "face-to-face",

"type": "practice",

"campus": "Кронверкский пр., д.49, лит.А",

"auditory": "2307",

"teacher": "Лучинкин Константин Сергеевич",

"time": {

"period": **8**,

"from\_time": "20:20",

"to\_time": "21:50"

}

}

]

}

## Решение основного задания

### Исходный код

#### Основная программа

**import** **os**

**def** **main**():

script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

json\_file\_path = os.path.join(script\_dir, 'schedule.json')

**with** open(json\_file\_path, 'r', encoding='utf-8') **as** f:

json\_string = f.read()

data = parse\_json(json\_string)

yaml\_str = "---**\n**" + to\_yaml(data)

print(yaml\_str)

main()

#### Токенайзер

WHITESPACE = (' ', '**\t**', '**\n**', '**\r**')

SYNTAX = {

'{': 'LEFT\_BRACE',

'}': 'RIGHT\_BRACE',

'[': 'LEFT\_BRACKET',

']': 'RIGHT\_BRACKET',

',': 'COMMA',

':': 'COLON'

}

**def** **tokenize**(s):

# превращает строку JSON в последовательность токенов

tokens = []

i = **0**

length = len(s)

**while** i < length:

char = s[i]

**if** char **in** WHITESPACE:

i += **1**

**continue**

**if** char **in** SYNTAX:

tokens.append(SYNTAX[char])

i += **1**

**continue**

**if** char == '"':

i += **1** # пропускаем саму кавычку

start = i

**while** i < length **and** s[i] != '"':

i += **1**

**if** i >= length:

**raise** **ValueError**("незавершенная строка")

string\_value = s[start:i]

tokens.append(f"STRING:{string\_value}")

i += **1** # пропускаем закрывающую кавычку

**continue**

**if** char.isdigit():

start = i

i += **1**

**while** i < length **and** s[i].isdigit():

i += **1**

number\_str = s[start:i]

tokens.append(f"NUMBER:{number\_str}")

**continue**

**if** s.startswith('true', i):

tokens.append("BOOL:true")

i += **4**

**continue**

**if** s.startswith('false', i):

tokens.append("BOOL:false")

i += **5**

**continue**

**if** s.startswith('null', i):

tokens.append("NULL")

i += **4**

**continue**

**raise** **ValueError**(f"неизвестный символ '{char}' на позиции {i}")

**return** tokens

#### Парсер JSON

**def** **parse\_json**(s):

# принимает строку JSON, токенизирует и парсит её,

# возвращая Python-объект (dict, list, str, int/float, bool, None)

list\_of\_tokens = tokenize(s)

result, remaining = parse\_tokens(list\_of\_tokens, **0**)

**if** remaining != len(list\_of\_tokens):

**raise** **ValueError**("остались непрочитанные токены")

**return** result

**def** **parse\_tokens**(tokens, i):

# разбирает список токенов и возвращает кортеж (полученный объект, новая\_позиция)

**if** i >= len(tokens):

**raise** **ValueError**("выход за границу списка токенов")

current = tokens[i]

**if** current == 'LEFT\_BRACE':

**return** parse\_object(tokens, i + **1**)

**elif** current == 'LEFT\_BRACKET':

**return** parse\_array(tokens, i + **1**)

**elif** current.startswith("STRING:"):

**return** current.split("STRING:", **1**)[**1**], i + **1**

**elif** current.startswith("NUMBER:"):

num\_str = current.split("NUMBER:", **1**)[**1**]

**return** int(num\_str), i + **1**

**elif** current.startswith("BOOL:"):

val\_str = current.split("BOOL:", **1**)[**1**]

**return** (**True** **if** val\_str == 'true' **else** **False**), i + **1**

**elif** current == 'NULL':

**return** **None**, i + **1**

**def** **parse\_object**(tokens, i):

# парсим объект (после '{')

obj = {}

# рассматриваем пустой объект {}

**if** i < len(tokens) **and** tokens[i] == 'RIGHT\_BRACE':

**return** obj, i + **1**

**while** **True**:

**if** i >= len(tokens):

**raise** **ValueError**("выход за границу списка токенов")

# Ожидаем STRING:key

key\_token = tokens[i]

**if** **not** key\_token.startswith("STRING:"):

**raise** **ValueError**(f"ожидалось слово, получили {key\_token}")

key = key\_token.split("STRING:", **1**)[**1**]

i += **1**

# Ожидаем ':'

**if** i >= len(tokens) **or** tokens[i] != 'COLON':

**raise** **ValueError**("ожидалось ':' после ключа")

i += **1** # пропускаем 'COLON'

# Теперь парсим значение

value, i = parse\_tokens(tokens, i)

obj[key] = value

# Либо конец объекта, либо запятая

**if** i >= len(tokens):

**raise** **ValueError**("выход за границу списка токенов")

**if** tokens[i] == 'RIGHT\_BRACE':

**return** obj, i + **1**

**elif** tokens[i] == 'COMMA':

i += **1**

**continue**

**else**:

**raise** **ValueError**(f"ожидалась ',' или '}}', получили {tokens[i]}")

**def** **parse\_array**(tokens, i):

arr = []

# Может быть пустой массив: [ ]

**if** i < len(tokens) **and** tokens[i] == 'RIGHT\_BRACKET':

**return** arr, i + **1**

**while** **True**:

**if** i >= len(tokens):

**raise** **ValueError**("выход за границу списка токенов")

# Парсим элемент

value, i = parse\_tokens(tokens, i)

arr.append(value)

**if** i >= len(tokens):

**raise** **ValueError**("выход за границу списка токенов")

**if** tokens[i] == 'RIGHT\_BRACKET':

**return** arr, i + **1**

**elif** tokens[i] == 'COMMA':

i += **1**

**continue**

**else**:

**raise** **ValueError**(f"ожидалась ',' или ']', получили {tokens[i]}")

#### Преобразование в YAML

**def** **format\_scalar**(value):

# преобразует простое значение (число, строку, bool, None) в строку

**if** value **is** **True**:

**return** "true"

**elif** value **is** **False**:

**return** "false"

**elif** value **is** **None**:

**return** "null"

**else**:

**return** str(value)

**def** **to\_yaml**(data, indent=**0**):

"""

рекурсивная функция, которая обходит Python-объект (dict, list, скаляр)

и формирует YAML-подобную строку

"""

**def** **spaces**(n):

**return** ' ' \* n

**if** isinstance(data, dict):

# Словарь

lines = []

**for** key, value **in** data.items():

**if** isinstance(value, (dict, list)):

# вложенная структура

lines.append(f"{spaces(indent)}{key}:")

lines.append(to\_yaml(value, indent + **2**))

**else**:

# Простой элемент в одну строку

lines.append(f"{spaces(indent)}{key}: {format\_scalar(value)}")

**return** '**\n**'.join(lines)

**elif** isinstance(data, list):

lines = []

**for** item **in** data:

**if** isinstance(item, (dict, list)):

lines.append(f"{spaces(indent)}-")

lines.append(to\_yaml(item, indent + **2**))

**else**:

lines.append(f"{spaces(indent)}- {format\_scalar(item)}")

**return** '**\n**'.join(lines)

**else**:

# Скаляр

**return** f"{spaces(indent)}{format\_scalar(data)}"

### Результат программы

**---**

even\_week: true

group\_name: P3118

date: Tuesday, 3 Dec 2024

schedule:

-

subject: English

class\_format: face-to-face

type: practice

campus: ул. Ломоносова 9

auditory: 3211

teacher: Ермакова Елена Владиславовна

time:

period: 1

from\_time: 08:20

to\_time: 09:50

-

subject: English

class\_format: face-to-face

type: practice

campus: ул. Ломоносова 9

auditory: 3211

teacher: Ермакова Елена Владиславовна

time:

period: 2

from\_time: 10:00

to\_time: 11:30

-

subject: Discrete\_Math

class\_format: face-to-face

type: lecture

campus: Кронверкский пр., д.49, лит.А

auditory: 1404

teacher: Карпов Дмитрий Валерьевич

time:

period: 7

from\_time: 18:40

to\_time: 20:10

-

subject: Discrete\_Math

class\_format: face-to-face

type: practice

campus: Кронверкский пр., д.49, лит.А

auditory: 2307

teacher: Лучинкин Константин Сергеевич

time:

period: 8

from\_time: 20:20

to\_time: 21:50

## Решение доп. задания 1

### Исходный код

**import** **os**

**import** **json**

**import** **yaml**

**def** **json\_to\_yaml**(json\_input, yaml\_output):

**with** open(json\_input, 'r', encoding='utf-8') **as** json\_file:

data = json.load(json\_file)

yaml\_data = yaml.dump(data, allow\_unicode=**True**, default\_flow\_style=**False**, sort\_keys=**False**)

**with** open(yaml\_output, 'w', encoding='utf-8') **as** yaml\_file:

yaml\_file.write('---**\n**')

yaml\_file.write(yaml\_data)

print(f"YAML сохранен в '{yaml\_output}'.")

**def** **main**():

script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

json\_file = os.path.join(script\_dir, 'schedule.json')

yaml\_file = os.path.join(script\_dir, 'schedule.yaml')

json\_to\_yaml(json\_file, yaml\_file)

main()

### Сравнение результатов

Сходства:

1. Оба вывода имеют одинаковую структуру, включая ключи и вложенные объекты.
2. Форматирование списков: В обоих случаях списки (schedule) представлены с дефисами (-) перед первыми ключами каждого элемента.
3. Булевые значения, строки, числа и null представлены аналогично.
4. В обоих выводах используются отступы для обозначения вложенности.

Различия:

1. Самодельный парсер предполагает, что строки уже корректно закодированы и выводятся как есть.PyYAML поддерживает Unicode и корректно отображает кириллические символы благодаря параметру allow\_unicode=True.
2. Самодельный парсер медленный и менее надежны, особенно при работе с большими или сложными JSON-файлами. PyYAML оптимизирован для высокой производительности и надежности, включая обработку ошибок и соответствие спецификациям YAML.
3. Готовые библиотеки корректно обрабатывают такие случаи, выводя [] для пустых списков и {} для пустых словарей, в то время как самодельный парсер может требовать дополнительной логики для этих случаев.

## Решение доп. задания 2

### Исходный код

**import** **os**

**import** **re**

WHITESPACE = re.compile(r'\s+')

SYNTAX = {

'{': 'LEFT\_BRACE',

'}': 'RIGHT\_BRACE',

'[': 'LEFT\_BRACKET',

']': 'RIGHT\_BRACKET',

',': 'COMMA',

':': 'COLON'

}

STRING\_REGEX = re.compile(r'^"([^"**\\**]\*)"')

NUMBER\_REGEX = re.compile(r'^\d+')

BOOL\_REGEX = re.compile(r'^(true|false)\b')

NULL\_REGEX = re.compile(r'^null\b')

**def** **tokenize**(s):

# Превращает строку JSON в последовательность токенов с использованием регулярных выражений.

tokens = []

i = **0**

length = len(s)

**while** i < length:

substring = s[i:]

ws\_match = WHITESPACE.match(substring)

**if** ws\_match:

i += ws\_match.end()

**continue**

char = substring[**0**]

**if** char **in** SYNTAX:

tokens.append(SYNTAX[char])

i += **1**

**continue**

string\_match = STRING\_REGEX.match(substring)

**if** string\_match:

string\_value = string\_match.group(**1**)

tokens.append(f"STRING:{string\_value}")

i += string\_match.end()

**continue**

number\_match = NUMBER\_REGEX.match(substring)

**if** number\_match:

number\_str = number\_match.group(**0**)

tokens.append(f"NUMBER:{number\_str}")

i += number\_match.end()

**continue**

bool\_match = BOOL\_REGEX.match(substring)

**if** bool\_match:

bool\_str = bool\_match.group(**1**)

tokens.append(f"BOOL:{bool\_str}")

i += bool\_match.end()

**continue**

null\_match = NULL\_REGEX.match(substring)

**if** null\_match:

tokens.append("NULL")

i += null\_match.end()

**continue**

**raise** **ValueError**(f"Неизвестный символ '{char}' на позиции {i}")

**return** tokens

**def** **parse\_json**(s):

# получает строку JSON -> токенизирует -> парсит -> возвращает Python-объект.

list\_of\_tokens = tokenize(s)

result, remaining = parse\_tokens(list\_of\_tokens, **0**)

**if** remaining != len(list\_of\_tokens):

**raise** **ValueError**("Остались непрочитанные токены")

**return** result

**def** **parse\_tokens**(tokens, i):

# Разбирает список токенов, начиная с позиции i и возвращает кортеж (полученный объект, новая позиция).

**if** i >= len(tokens):

**raise** **ValueError**("Выход за границу списка токенов")

current = tokens[i]

**if** current == 'LEFT\_BRACE':

**return** parse\_object(tokens, i + **1**)

**elif** current == 'LEFT\_BRACKET':

**return** parse\_array(tokens, i + **1**)

**elif** current.startswith("STRING:"):

**return** current.split("STRING:", **1**)[**1**], i + **1**

**elif** current.startswith("NUMBER:"):

num\_str = current.split("NUMBER:", **1**)[**1**]

**if** '.' **in** num\_str **or** 'e' **in** num\_str.lower():

**return** float(num\_str), i + **1**

**else**:

**return** int(num\_str), i + **1**

**elif** current.startswith("BOOL:"):

val\_str = current.split("BOOL:", **1**)[**1**]

**return** (**True** **if** val\_str == 'true' **else** **False**), i + **1**

**elif** current == 'NULL':

**return** **None**, i + **1**

**else**:

**raise** **ValueError**(f"Неожиданный токен '{current}' на позиции {i}")

**def** **parse\_object**(tokens, i):

# Парсим объект (после '{').

obj = {}

# Рассматриваем пустой объект {}

**if** i < len(tokens) **and** tokens[i] == 'RIGHT\_BRACE':

**return** obj, i + **1**

**while** **True**:

**if** i >= len(tokens):

**raise** **ValueError**("Выход за границу списка токенов")

# Ожидаем STRING:key

key\_token = tokens[i]

**if** **not** key\_token.startswith("STRING:"):

**raise** **ValueError**(f"Ожидалась строка, получили {key\_token}")

key = key\_token.split("STRING:", **1**)[**1**]

i += **1**

# Ожидаем ':'

**if** i >= len(tokens) **or** tokens[i] != 'COLON':

**raise** **ValueError**("Ожидалось ':' после ключа")

i += **1** # пропускаем 'COLON'

value, i = parse\_tokens(tokens, i)

obj[key] = value

# Либо конец объекта, либо запятая

**if** i >= len(tokens):

**raise** **ValueError**("Выход за границу списка токенов при парсинге объекта")

**if** tokens[i] == 'RIGHT\_BRACE':

**return** obj, i + **1**

**elif** tokens[i] == 'COMMA':

i += **1**

**continue**

**else**:

**raise** **ValueError**(f"Ожидалась ',' или '}}', получили {tokens[i]}")

**def** **parse\_array**(tokens, i):

# Парсим массив (после '[').

arr = []

**if** i < len(tokens) **and** tokens[i] == 'RIGHT\_BRACKET':

**return** arr, i + **1**

**while** **True**:

**if** i >= len(tokens):

**raise** **ValueError**("Выход за границу списка токенов при парсинге массива")

value, i = parse\_tokens(tokens, i)

arr.append(value)

**if** i >= len(tokens):

**raise** **ValueError**("Выход за границу списка токенов при парсинге массива")

**if** tokens[i] == 'RIGHT\_BRACKET':

**return** arr, i + **1**

**elif** tokens[i] == 'COMMA':

i += **1**

**continue**

**else**:

**raise** **ValueError**(f"Ожидалась ',' или ']', получили {tokens[i]}")

**def** **format\_scalar**(value):

**if** value **is** **True**:

**return** "true"

**elif** value **is** **False**:

**return** "false"

**elif** value **is** **None**:

**return** "null"

**else**:

# Если строка содержит специальные символы, пробелы или начинается с цифры, оборачиваем в кавычки

**if** isinstance(value, str):

**if** re.search(r'[,:{}\[\]#&\*!|>**\'**"`]', value) **or** re.match(r'^\d', value):

# Экранируем внутренние кавычки

escaped = value.replace('"', '**\\**"')

**return** f'"{escaped}"'

**return** str(value)

**def** **to\_yaml**(data, indent=**0**):

**def** **spaces**(n):

**return** ' ' \* n

**if** isinstance(data, dict):

lines = []

**for** key, value **in** data.items():

**if** isinstance(value, (dict, list)):

# Вложенная структура

lines.append(f"{spaces(indent)}{key}:")

lines.append(to\_yaml(value, indent + **2**))

**else**:

# Простой элемент в одну строку

lines.append(f"{spaces(indent)}{key}: {format\_scalar(value)}")

**return** '**\n**'.join(lines)

**elif** isinstance(data, list):

lines = []

**for** item **in** data:

**if** isinstance(item, dict):

# Ищем первый ключ для компактного вывода с дефисом

item\_keys = list(item.keys())

**if** len(item\_keys) > **0**:

first\_key = item\_keys[**0**]

first\_value = item[first\_key]

# Формируем строку с дефисом и первым ключом

line = f"{spaces(indent)}- {first\_key}: {format\_scalar(first\_value)}"

lines.append(line)

# Создаём копию словаря без первого ключа

temp\_dict = dict(item)

temp\_dict.pop(first\_key)

**if** len(temp\_dict) > **0**:

# Рекурсивно обрабатываем оставшиеся ключи с увеличенным отступом

sub\_yaml = to\_yaml(temp\_dict, indent + **2**)

lines.append(sub\_yaml)

**else**:

# Пустой словарь

lines.append(f"{spaces(indent)}- {{}}")

**elif** isinstance(item, list):

# Вложенный список

lines.append(f"{spaces(indent)}-")

lines.append(to\_yaml(item, indent + **2**))

**else**:

# Скалярное значение

lines.append(f"{spaces(indent)}- {format\_scalar(item)}")

**return** '**\n**'.join(lines)

**else**:

# Скаляр

**return** f"{spaces(indent)}{format\_scalar(data)}"

**def** **main**():

script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

json\_file\_path = os.path.join(script\_dir, 'schedule.json')

**with** open(json\_file\_path, 'r', encoding='utf-8') **as** f:

json\_string = f.read()

**try**:

data = parse\_json(json\_string)

**except** **ValueError** **as** ve:

print(f"Ошибка при парсинге JSON: {ve}")

**return**

yaml\_str = "---**\n**" + to\_yaml(data)

print(yaml\_str)

main()

### Сравнение результатов

С регулярными выражениями токенизация стала более компактной и эффективной за счёт использования заранее определённых паттернов для различных типов токенов. Код стал более компактным, читаемым и легко расширяемым при необходимости добавления новых правил токенизации.

# Заключение:

# Список использованных источников:

1. Лямин А.В., Череповская Е.Н. Объектно-ориентированное программирование. Компьютерный практикум. – СПб: Университет ИТМО, 2017. – 143 с.