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»**

по дисциплине «Информатика»

Вариант № 12

**Группа:** P3112

**Выполнил**: Пархоменко К. А.

**Проверил:** Белозубов А.В.

г. Санкт-Петербург

2022 г.

Оглавление

[Вариант задания 3](#_Toc120798875)

[Задания 3](#_Toc120798876)

[1. Обязательное задание 3](#_Toc120798877)

[2. Дополнительное задание №1 3](#_Toc120798878)

[3. Дополнительное задание №2 3](#_Toc120798879)

[4. Дополнительное задание №3 3](#_Toc120798880)

[5. Дополнительное задание №4 3](#_Toc120798881)

[Список литературы 3](#_Toc120798882)

# Вариант задания

|  |  |  |  |
| --- | --- | --- | --- |
| **№ Варианта** | **Исходный формат** | **Результирующий формат** | **День недели** |
| 12 | JSON | XML | Среда |

Задания

# Обязательное задание

1. Написать программу на языке Python 3.x, которая бы осуществляла парсинг и конвертацию исходного файла в новый.

import sys

PADDING = " "

def fix(some\_string: str) -> str:

"""

Замена спецсимволов

:param some\_string: строка

:return: измененная строка

"""

return some\_string.replace("&", "&amp;").replace('"', "&quot;") \

.replace("'", "&apos;").replace("<", "&lt;") \

.replace(">", "&gt;").replace(r"\\", "")

def fix\_tag(tag: str) -> str:

"""

Замена спецсимволов в тегах

:param tag: тег

:return: исправленный тег

"""

return tag.replace(' ', '\_').replace("\\", "") \

.replace('"', '').replace(";", "\_").replace("<", "\_lt\_") \

.replace(">", "\_gt\_").replace("&", "\_amp\_").replace("'", "\_apos\_") \

.replace("=", "\_eq\_").replace(":", "\_colon\_").replace("|", "\_vl\_").replace("\"", "\_quot\_")

def converter(json\_object: str, padding: str = " ") -> str:

result\_list = []

json\_obj\_type = type(json\_object)

if json\_obj\_type is list:

# Парсинг списка

for list\_element in json\_object:

result\_list.append(converter(list\_element, padding))

return "\n".join(result\_list)

if json\_obj\_type is dict:

# Парсинг тегов

for tag in json\_object.keys():

xml\_tag = fix\_tag(tag)

result\_list.append(padding + f'<{xml\_tag}>')

# Вызываем функцию рекурсивно, чтобы учесть данные между тегами

result\_list.append(converter(json\_object[tag], PADDING + padding))

result\_list.append(padding + f"</{xml\_tag}>")

return "\n".join(result\_list)

return padding + fix(json\_object)

with open('Tests/test.json', 'r', encoding='utf-8') as json\_file:

data = json\_file.read()

try:

dic = eval(data)

except SyntaxError as error:

print("error: bad json. " + error.msg)

sys.exit(1)

with open(f'{json\_file.name.replace(".json", "")}.xml', 'w', encoding='utf-8') as xml:

xml.write('<?xml version="1.0" encoding="UTF-8"?>\n')

xml.write('<Расписание>\n')

xml.write(converter(dic, PADDING) + "\n")

xml.write('</Расписание>')

xml.close()

Код программы для задания № 1

# 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.

Результат конвертации с помощью библиотеки json2xml очень похож, на результат, который выдал мой конвертер. Главное отличие заключается в том, что библиотека указывает тип данных для каждого XML тэга.

import json2xml.json2xml

from json2xml.utils import readfromstring

with open("Tests/test.json", "r", encoding='utf-8') as json\_file:

xml = open(f'{json\_file.name.replace(".json", "")}\_lib.xml', 'w', encoding='utf-8')

data = readfromstring(json\_file.read())

xml.write(json2xml.json2xml.Json2xml(data, wrapper="all", pretty=True).to\_xml())

xml.close()

Код программы для задания № 2

# Дополнительное задание №2

1. Переписать исходный код, добавив в него использование регулярных выражений.
2. Сравнить полученные результаты и объяснить их сходство/различие.

Результат, полученный с помощью переписанного кода, полностью идентичен результату, полученному конвертером из первого задания.  
Все символы, которые могут создать не валидный XML файл, также заменены.

import re

import sys

TEXT\_PATTERN = re.compile(r"(.+)")

FILE\_PATTERN = re.compile(".json$")

PADDING = " "

def fix(some\_str: str) -> str:

some\_str = re.sub("&", "&amp;", some\_str)

some\_str = re.sub('"', "&quot;", some\_str)

some\_str = re.sub("'", "&apos;", some\_str)

some\_str = re.sub("<", "&lt;", some\_str)

some\_str = re.sub(">", "&gt;", some\_str)

some\_str = re.sub(r"\\", "", some\_str)

return some\_str

def fix\_tag(tag: str) -> str:

tag = re.sub(" ", "\_", tag)

tag = re.sub(r'\\', "", tag)

tag = re.sub(";", "\_", tag)

tag = re.sub("<", "\_lt\_", tag)

tag = re.sub("\"", "\_quot\_", tag)

tag = re.sub(":", "\_colon\_", tag)

tag = re.sub(">", "\_gt\_", tag)

tag = re.sub("'", "\_apos\_", tag)

tag = re.sub("\"", "\_\_", tag)

tag = re.sub("=", "\_eq\_", tag)

tag = re.sub("&", "\_amp\_", tag)

tag = re.sub(r"\|", "\_vl\_", tag)

return tag

def converter(json\_object, padding=" "):

result\_list = []

json\_obj\_type = type(json\_object)

if json\_obj\_type is list:

# Парсинг списка

for list\_element in json\_object:

result\_list.append(converter(list\_element, padding))

return "\n".join(result\_list)

if json\_obj\_type is dict:

# Парсинг тегов

for tag in json\_object.keys():

xml\_tag = fix\_tag(tag)

result\_list.append(TEXT\_PATTERN.sub(fr"{padding}<\1>", xml\_tag))

# Вызываем функцию рекурсивно, чтобы учесть данные между тегами

result\_list.append(converter(json\_object[tag], PADDING + padding))

result\_list.append(TEXT\_PATTERN.sub(fr"{padding}</\1>", xml\_tag))

return "\n".join(result\_list)

return padding + fix(json\_object)

with open('Tests/test.json', 'r', encoding='utf-8') as json\_file:

data = json\_file.read()

try:

dic = eval(data)

except (SyntaxError, TypeError) as error:

print("error: bad json. " + error.msg)

sys.exit(1)

with open(re.sub(".json$", "\_re.xml", json\_file.name), 'w', encoding='utf-8') as xml:

xml.write('<?xml version="1.0" encoding="UTF-8"?>\n')

xml.write('<Расписание>\n')

xml.write(converter(dic, PADDING) + "\n")

xml.write('</Расписание>')

xml.close()

Код программы для задания № 3

# Дополнительное задание №3

1. Используя свою исходную программу из обязательного задания, программу из дополнительного задания №1 и программу из дополнительного задания №2, сравнить стократное время выполнения парсинга + конвертации в цикле.
2. Проанализировать полученные результаты и объяснить их сходство/различие.

Опираясь на полученные данные, можно сделать вывод, что алгоритм, использующийся в первом задании – самый быстрый, а программа, использующая библиотеку для конвертации, является самой медленной.

:::::TUSK #1 BENCHMARK:::::

Benchmark result: 0.029129499991540797

:::::TUSK #2 BENCHMARK:::::

Benchmark result: 0.057906300004106015

:::::TUSK #3 BENCHMARK:::::

Benchmark result: 0.16033969999989495

import sys

import timeit

import json2xml.json2xml

from json2xml.utils import readfromstring

import Lab4\_default

import Lab4\_with\_regex

def test\_1():

padding = " "

with open('Tests/test.json', 'r', encoding='utf-8') as json\_file:

data = json\_file.read()

try:

dic = eval(data)

except (SyntaxError, TypeError) as error:

print("error: bad json. " + error.msg)

sys.exit(1)

with open(f'{json\_file.name.replace(".json", "")}.xml', 'w', encoding='utf-8') as xml:

xml.write('<?xml version="1.0" encoding="UTF-8"?>\n')

xml.write('<Расписание>\n')

xml.write(Lab4\_default.converter(dic, padding) + "\n")

xml.write('</Расписание>')

xml.close()

def test\_2():

padding = " "

with open('Tests/test.json', 'r', encoding='utf-8') as json\_file:

data = json\_file.read()

try:

dic = eval(data)

except (SyntaxError, TypeError) as error:

print("error: bad json. " + error.msg)

sys.exit(1)

with open(Lab4\_with\_regex.FILE\_PATTERN.sub("\_re.xml", json\_file.name), 'w', encoding='utf-8') as xml:

xml.write('<?xml version="1.0" encoding="UTF-8"?>\n')

xml.write('<Расписание>\n')

xml.write(Lab4\_with\_regex.converter(dic, padding) + "\n")

xml.write('</Расписание>')

xml.close()

def test\_3():

with open("Tests/test.json", "r", encoding='utf-8') as json\_file:

with open(f'{json\_file.name.replace(".json", "")}\_lib.xml', 'w', encoding='utf-8') as xml:

data = readfromstring(json\_file.read())

xml.write(json2xml.json2xml.Json2xml(data, wrapper="all", pretty=True).to\_xml())

xml.close()

if \_\_name\_\_ == '\_\_main\_\_':

# number of cycles

N = 100

print(":::::TUSK #1 BENCHMARK:::::")

print(f"Benchmark result: {timeit.timeit('test\_1()', number=N, globals=locals())}\n")

print(":::::TUSK #2 BENCHMARK:::::")

print(f"Benchmark result: {timeit.timeit('test\_2()', number=N, globals=locals())}\n")

print(":::::TUSK #3 BENCHMARK:::::")

print(f"Benchmark result: {timeit.timeit('test\_3()', number=N, globals=locals())}\n")

Код программы для задания № 4

# 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.

Формат CSV представляет табличные данные в текстовом формате.

import sys

from pandas import read\_json

with open("Tests/test.json", "r", encoding="utf-8") as json\_file:

try:

pd\_json = read\_json(json\_file)

except ValueError as e:

print("error: bad json.")

sys.exit(1)

pd\_json.to\_csv(f'{json\_file.name.replace(".json", "")}.csv', encoding='utf-8')

Код программы для задания № 4

# Список литературы

**XML для начинающих** [В Интернете] / авт. Microsoft // support.microsoft.com. - 22 Ноябрь 2022 г.. - https://support.microsoft.com/ru-ru/office/xml-%D0%B4%D0%BB%D1%8F-%D0%BD%D0%B0%D1%87%D0%B8%D0%BD%D0%B0%D1%8E%D1%89%D0%B8%D1%85-a87d234d-4c2e-4409-9cbc-45e4eb857d44.

**Форма Бэкуса — Наура** [В Интернете] // Wikipedia. - 15 Сентябрь 2021 г.. - 22 Ноябрь 2022 г.. - https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0\_%D0%91%D1%8D%D0%BA%D1%83%D1%81%D0%B0\_%E2%80%94\_%D0%9D%D0%B0%D1%83%D1%80%D0%B0.