Федеральное государственное автономное образовательное учреждение высшего образования «Научно-образовательная корпорация ИТМО»

Факультет Программной Инженерии и Компьютерной Техники

**Лабораторная работа по информатике №4**

вариант: 14

Преподаватель: Рудникова Тамара Владимировна

Выполнил: Степанов Арсений Алексеевич

Группа: Р3109

**Оглавление**

[Цели 3](#_Toc120639387)

[Исходный JSON 3](#_Toc120639388)

[Задание 1 3](#_Toc120639389)

[Код программы 3](#_Toc120639390)

[Полученный YML 10](#_Toc120639391)

[Задание 2 11](#_Toc120639392)

[Код программы 11](#_Toc120639393)

[Полученный YML 11](#_Toc120639394)

[Задание 3 12](#_Toc120639395)

[Код программы 12](#_Toc120639396)

[Полученный YML 16](#_Toc120639397)

[Задание 4 16](#_Toc120639398)

[Итоги 17](#_Toc120639399)

# Цели

Научиться работать с различными форматами представления данных, языками разметки используемыми для этого, а также различными библиотеками, предназначенных для конвертации этих данных из одного формата в другой.

# Исходный JSON

{

    "lectures":

    [

        {

            "type": "Лабораторная",

            "form": "Очно - дистанционный",

            "teacher": "Рудникова Тамара Владимировна",

            "auditory": "2305/1 (БЫВШ. 303/2 (УСЛ)) АУД.",

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

            "start": "11:40",

            "end": "13:10",

            "weeks": [2, 4, 6, 8, 10, 12, 14, 16]

        },

        {

            "type": "Лабораторная",

            "form": "Очно - дистанционный",

            "teacher": "Рудникова Тамара Владимировна",

            "auditory": "2305/1 (БЫВШ. 303/2 (УСЛ)) АУД.",

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

            "start": "13:30",

            "end": "15:00",

            "weeks": [2, 4, 6, 8, 10, 12, 14, 16]

        }

    ]

}

# Задание 1

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

## Код программы

# By Armemius

import time

DEBUG: bool = False

pos: int = 0

size: int = 0

class TreeNode:

    def \_\_init\_\_(self, name: str, type: str, content: str = ""):

        self.name = name

        self.type = type

        self.content = content

        self.nodes = []

    def addNode(self, node):

        self.nodes.append(node)

def printTree(node: TreeNode, dep: int = 0):

    print("| " \* dep + "+", f"Id: {node.name}, type: {node.type}" + (f", content: {node.content}" if node.content != "" else ""))

    for it in node.nodes:

        printTree(it, dep + 1)

# UTIL

def next():

    global pos, size

    pos += 1

    if pos == size + 1:

        raise RuntimeError("Met end of file while parsing")

def removeSpaces(text: str):

    res = ""

    qFlag = False

    sFlag = False

    for it in text:

        if it == '"' and not sFlag:

            qFlag = qFlag != True

        if it == "\n" and qFlag:

            raise RuntimeError(f"Unexpexted end of line at pos: {pos}")

        if it not in " \n\t\r":

            res += it

            if it == "\\":

                sFlag = sFlag != True

                continue

        elif qFlag:

            res += it

        sFlag = False

    return res

# PARSING

def parse(name: str, text: str) -> TreeNode:

    global pos

    if (DEBUG): print(f"'parse' function called. Params: pos - {pos}, curr char - {text[pos]}")

    if (text[pos] == "{"):

        return parseObject(name, text)

    elif (text[pos] in "0123456789"):

        return parseNumber(name, text)

    elif (text[pos] == "\""):

        return parseString(name, text)

    elif (text[pos] == "["):

        return parseArray(name, text)

    elif (text[pos] in "tf"):

        return parseBoolean(name, text)

    elif (text[pos] == "n"):

        return parseNull(name, text)

    raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}")

def parseField(text: str) -> TreeNode:

    global pos

    if (DEBUG): print(f"'parseField' function called. Params: pos - {pos}, curr char - {text[pos]}")

    if text[pos] != '"':

        raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, expected '\"'")

    return parse(parseId(text), text)

def parseId(text: str) -> str:

    global pos

    if (DEBUG): print(f"'parseId' function called. Params: pos - {pos}, curr char - {text[pos]}")

    next()

    content = ""

    sFlag = False

    while text[pos] != '"' or sFlag:

        if text[pos] == "\\":

            if sFlag == False:

                next()

                sFlag = True

                continue

            sFlag = False

        else:

            sFlag = False

        content += text[pos]

        next()

    next()

    if text[pos] != ':':

        raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, expected ':'")

    next()

    return content

def parseNumber(name: str, text: str) -> TreeNode:

    global pos

    if (DEBUG): print(f"'parseNumber' function called. Params: pos - {pos}, curr char - {text[pos]}")

# dotPos values:

# -2 -> flag for first char (if met dot on first char -> throw exception)

# -1 -> dot haven't met yet,

# else -> pos of located dot

    dotPos = -2

    content = ""

    while text[pos] in "0123456789.":

        if text[pos] == ".":

            if dotPos == -2:

                raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, '.' can't be first symbol in numeric value")

            elif dotPos == -1:

                dotPos = pos

            else:

                raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, multiple '.' in numeric value")

        content += text[pos]

        next()

        if (dotPos == -2):

            dotPos = -1

    if pos - dotPos == 1:

        raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, '.' can't be last symbol in numeric value")

    return TreeNode(name, "NUMBER", content)

def parseString(name: str, text: str) -> TreeNode:

    global pos

    if (DEBUG): print(f"'parseString' function called. Params: pos - {pos}, curr char - {text[pos]}")

    next()

    content = ""

    sFlag = False

    while text[pos] != '"' or sFlag:

        if text[pos] == "\\":

            if sFlag == False:

                next()

                sFlag = True

                continue

            sFlag = False

        else:

            sFlag = False

        content += text[pos]

        next()

    next()

    return TreeNode(name, "STRING", content)

def parseArray(name: str, text: str) -> TreeNode:

    global pos

    if (DEBUG): print(f"'parseArray' function called. Params: pos - {pos}, curr char - {text[pos]}")

    arrNode = TreeNode(name, "ARRAY")

    next()

    while text[pos] != "]":

        arrNode.addNode(parse("NULL", text))

        if text[pos] != "," and text[pos] != "]":

            raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, expected ',' or ']'")

        elif text[pos] == ",":

            next()

    next()

    return arrNode

def parseObject(name: str, text: str) -> TreeNode:

    global pos

    if (DEBUG): print(f"'parseObject' function called. Params: pos - {pos}, curr char - {text[pos]}")

    objNode = TreeNode(name, "OBJECT")

    next()

    while text[pos] != "}":

        objNode.addNode(parseField(text))

        if text[pos] != "," and text[pos] != "}":

            raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, expected ',' or " + "'}'")

        elif text[pos] == ",":

            next()

    next()

    return objNode

def parseBoolean(name: str, text: str) -> TreeNode:

    global pos

    if (DEBUG): print(f"'parseBoolean' function called. Params: pos - {pos}, curr char - {text[pos]}")

    if (text[str] == "t"):

        next()

        if (text[str] == "r"):

            next()

            if (text[str] == "u"):

                next()

                if (text[str] == "e"):

                    next()

                    return TreeNode(name, "BOOLEAN", "true")

                else: raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, expected 'e'")

            else: raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, expected 'u'")

        else: raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, expected 'r'")

    else:

        next()

        if (text[str] == "a"):

            next()

            if (text[str] == "l"):

                next()

                if (text[str] == "s"):

                    next()

                    if (text[str] == "e"):

                        next()

                        return TreeNode(name, "BOOLEAN", "false")

                    else: raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, expected 'e'")

                else: raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, expected 's'")

            else: raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, expected 'l'")

        else: raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, expected 'a'")

def parseNull(name: str, text: str) -> TreeNode:

    global pos

    if (DEBUG): print(f"'parseNull' function called. Params: pos - {pos}, curr char - {text[pos]}")

    if (text[str] == "n"):

        next()

        if (text[str] == "u"):

            next()

            if (text[str] == "l"):

                next()

                if (text[str] == "l"):

                    next()

                    return TreeNode(name, "NULL", "null")

                else: raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, expected 'l'")

            else: raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, expected 'l'")

        else: raise RuntimeError(f"Unexpexted symbol '{text[pos]}' at pos: {pos}, expected 'u'")

def parseJson(filename: str) -> TreeNode:

    global pos, size

    pos = 0

    file = open(filename, "r")

    raw = removeSpaces(file.read())

    file.close()

    if DEBUG:

        print(raw)

    size = len(raw)

    return parseObject("JSON", raw)

# GENERATION

def genNumber(node: TreeNode, dep: int, arr: bool = False) -> str:

    if arr:

        return "  " \* dep + f"- {node.content}\n"

    return "  " \* dep + f"\"{node.name}\": {node.content}\n"

def genString(node: TreeNode, dep: int, arr: bool = False) -> str:

    if arr:

        return "  " \* dep + f"- \"{node.content}\"\n"

    return "  " \* dep + f"\"{node.name}\": \"{node.content}\"\n"

def genObject(node: TreeNode, dep: int, arr: bool = False) -> str:

    res = ""

    if arr:

        res = "  " \* dep + f"-\n"

    else:

        res = "  " \* dep + f"\"{node.name}\": \n"

    for it in node.nodes:

        res += genFragment(it, dep + 1)

    return res

def genArray(node: TreeNode, dep: int, arr: bool = False) -> str:

    res = ""

    if arr:

        res = "  " \* dep + f"-\n"

    else:

        res = "  " \* dep + f"\"{node.name}\": \n"

    for it in node.nodes:

        res += genFragment(it, dep + 1, True)

    return res

def genBoolean(node: TreeNode, dep: int, arr: bool = False) -> str:

    if arr:

        return "  " \* dep + f"- {node.content}\n"

    return "  " \* dep + f"\"{node.name}\": {node.content}\n"

def genNull(node: TreeNode, dep: int, arr: bool = False) -> str:

    if arr:

        return "  " \* dep + f"- {node.content}\n"

    return "  " \* dep + f"\"{node.name}\": {node.content}\n"

def genFragment(node: TreeNode, dep: int, arr: bool = False) -> str:

    if node.type == "NUMBER":

        return genNumber(node, dep, arr)

    if node.type == "STRING":

        return genString(node, dep, arr)

    if node.type == "OBJECT":

        return genObject(node, dep, arr)

    if node.type == "ARRAY":

        return genArray(node, dep, arr)

    if node.type == "BOOLEAN":

        return genBoolean(node, dep, arr)

    return genNull(node, dep, arr)

def genYaml(root: TreeNode) -> str:

    res = "---\n"

    for it in root.nodes:

        res += genFragment(it, 0)

    return res

# EXECUTION

def main():

    root = parseJson("C:\\My Files\\Pituhon\\lab4\\test.json")

    if DEBUG:

        printTree(root)

    yml = genYaml(root)

    output = open("C:\\My Files\\Pituhon\\lab4\\out1.yml", "w")

    output.write(yml)

    output.close()

    return 0

start = time.time()

for it in range(100):

    main()

print(time.time() - start)

## Полученный YML

---

"lectures":

  -

    "type": "Лабораторная"

    "form": "Очно - дистанционный"

    "teacher": "Рудникова Тамара Владимировна"

    "auditory": "2305/1 (БЫВШ. 303/2 (УСЛ)) АУД."

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

    "start": "11:40"

    "end": "13:10"

    "weeks":

      - 2

      - 4

      - 6

      - 8

      - 10

      - 12

      - 14

      - 16

  -

    "type": "Лабораторная"

    "form": "Очно - дистанционный"

    "teacher": "Рудникова Тамара Владимировна"

    "auditory": "2305/1 (БЫВШ. 303/2 (УСЛ)) АУД."

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

    "start": "13:30"

    "end": "15:00"

    "weeks":

      - 2

      - 4

      - 6

      - 8

      - 10

      - 12

      - 14

      - 16

# Задание 2

Найти готовые библиотеки, осуществляющие аналогичный парсинг и конвертацию файлов. Переписать исходный код, применив найденные библиотеки.

## Код программы

import time

import yaml

import json

def main():

    with open('C:\\My Files\\Pituhon\\lab4\\test.json', 'r') as file:

        configuration = json.load(file)

    with open('C:\\My Files\\Pituhon\\lab4\\out2.yml', 'w') as yaml\_file:

        yaml.dump(configuration, yaml\_file)

start = time.time()

for it in range(100):

    main()

print(time.time() - start)

## Полученный YML

lectures:

- auditory: 2305/1 (БЫВШ. 303/2 (УСЛ)) АУД."

  end: '13:10'

  form: "Очно - дистанционный"

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

  start: '11:40'

  teacher: "Рудникова Тамара Владимировна"

  type: "Лабораторная"

  weeks:

  - 2

  - 4

  - 6

  - 8

  - 10

  - 12

  - 14

  - 16

- auditory: 2305/1 (БЫВШ. 303/2 (УСЛ)) АУД."

  end: '15:00'

  form: "Очно - дистанционный"

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

  start: '13:30'

  teacher: "Рудникова Тамара Владимировна"

  type: "Лабораторная"

  weeks:

  - 2

  - 4

  - 6

  - 8

  - 10

  - 12

  - 14

  - 16

# Задание 3

Переписать исходный код, добавив в него использование регулярных выражений.

## Код программы

# By Armemius

import re

import time

class TreeNode:

    def \_\_init\_\_(self, name: str, type: str, content: str = ""):

        self.name = name

        self.type = type

        self.content = content

        self.nodes = []

    def addNode(self, node):

        self.nodes.append(node)

def printTree(node: TreeNode, dep: int = 0):

    print("| " \* dep + "+", f"Id: {node.name}, type: {node.type}" + (f", content: {node.content}" if node.content != "" else ""))

    for it in node.nodes:

        printTree(it, dep + 1)

# UTIL

def next():

    global pos, size

    pos += 1

    if pos == size + 1:

        raise RuntimeError("Met end of file while parsing")

def getCurrValue():

    global pos, tokens

    return tokens[pos][0]

def getCurrToken():

    global pos, tokens

    return tokens[pos][1]

# PARSING

def parseJson() -> TreeNode:

    global pos, tokens

    return parseObject("JSON")

def parseField() -> TreeNode:

    str = getCurrValue()

    next()

    next()

    return parse(str)

def parse(name: str) -> TreeNode:

    if (getCurrToken() == "OBJ\_BEGIN"):

        return parseObject(name)

    elif (getCurrToken() == "NUMBER"):

        objNode = TreeNode(name, "NUMBER", getCurrValue())

        next()

        return objNode

    elif (getCurrToken() == "STRING"):

        objNode = TreeNode(name, "STRING", getCurrValue())

        next()

        return objNode

    elif (getCurrToken() == "ARR\_BEGIN"):

        return parseArray(name)

    elif (getCurrToken() == "BOOLEAN"):

        objNode = TreeNode(name, "BOOLEAN", getCurrValue())

        next()

        return objNode

    elif (getCurrToken() == "NULL"):

        objNode = TreeNode(name, "NULL", getCurrValue())

        next()

        return objNode

def parseObject(name: str) -> TreeNode:

    objNode = TreeNode(name, "OBJECT")

    next()

    while (getCurrToken() != "OBJ\_END"):

        objNode.addNode(parseField())

    next()

    return objNode

def parseArray(name: str) -> TreeNode:

    objNode = TreeNode(name, "ARRAY")

    next()

    while (getCurrToken() != "ARR\_END"):

        objNode.addNode(parse("NULL"))

    next()

    return objNode

# GENERATION

def genNumber(node: TreeNode, dep: int, arr: bool = False) -> str:

    if arr:

        return "  " \* dep + f"- {node.content}\n"

    return "  " \* dep + f"{node.name}: {node.content}\n"

def genString(node: TreeNode, dep: int, arr: bool = False) -> str:

    if arr:

        return "  " \* dep + f"- {node.content}\n"

    return "  " \* dep + f"{node.name}: {node.content}\n"

def genObject(node: TreeNode, dep: int, arr: bool = False) -> str:

    res = ""

    if arr:

        res = "  " \* dep + f"-\n"

    else:

        res = "  " \* dep + f"{node.name}: \n"

    for it in node.nodes:

        res += genFragment(it, dep + 1)

    return res

def genArray(node: TreeNode, dep: int, arr: bool = False) -> str:

    res = ""

    if arr:

        res = "  " \* dep + f"-\n"

    else:

        res = "  " \* dep + f"{node.name}: \n"

    for it in node.nodes:

        res += genFragment(it, dep + 1, True)

    return res

def genBoolean(node: TreeNode, dep: int, arr: bool = False) -> str:

    if arr:

        return "  " \* dep + f"- {node.content}\n"

    return "  " \* dep + f"\"{node.name}\": {node.content}\n"

def genNull(node: TreeNode, dep: int, arr: bool = False) -> str:

    if arr:

        return "  " \* dep + f"- {node.content}\n"

    return "  " \* dep + f"\"{node.name}\": {node.content}\n"

def genFragment(node: TreeNode, dep: int, arr: bool = False) -> str:

    if node.type == "NUMBER":

        return genNumber(node, dep, arr)

    if node.type == "STRING":

        return genString(node, dep, arr)

    if node.type == "OBJECT":

        return genObject(node, dep, arr)

    if node.type == "ARRAY":

        return genArray(node, dep, arr)

    if node.type == "BOOLEAN":

        return genBoolean(node, dep, arr)

    return genNull(node, dep, arr)

def genYaml(root: TreeNode) -> str:

    res = "---\n"

    for it in root.nodes:

        res += genFragment(it, 0)

    return res

# EXECUTION

def main():

    global tokens, pos, size

    file = open("C:\\My Files\\Pituhon\\lab4\\test.json", "r")

    raw = file.read()

    pt = re.compile(r"(?P<STRING>\".+?\")|(?P<COLON>:)|(?P<OBJ\_BEGIN>{)|(?P<OBJ\_END>})|(?P<ARR\_BEGIN>\[)|(?P<ARR\_END>\])|(?P<BOOLEAN>true|false)|(?P<NULL>null)|(?P<NUMBER>\d+(\.\d+)?)")

    tokens = [(it[0], it.lastgroup) for it in re.finditer(pt, raw)]

    pos = 0

    size = len(tokens)

    root = parseJson()

    yml = genYaml(root)

    output = open("C:\\My Files\\Pituhon\\lab4\\out3.yml", "w")

    output.write(yml)

    output.close()

    return 0

start = time.time()

for it in range(100):

    main()

print(time.time() - start)

## Полученный YML

---

"lectures":

  -

    "type": "Лабораторная"

    "form": "Очно - дистанционный"

    "teacher": "Рудникова Тамара Владимировна"

    "auditory": "2305/1 (БЫВШ. 303/2 (УСЛ)) АУД."

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

    "start": "11:40"

    "end": "13:10"

    "weeks":

      - 2

      - 4

      - 6

      - 8

      - 10

      - 12

      - 14

      - 16

  -

    "type": "Лабораторная"

    "form": "Очно - дистанционный"

    "teacher": "Рудникова Тамара Владимировна"

    "auditory": "2305/1 (БЫВШ. 303/2 (УСЛ)) АУД."

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

    "start": "13:30"

    "end": "15:00"

    "weeks":

      - 2

      - 4

      - 6

      - 8

      - 10

      - 12

      - 14

      - 16

# Задание 4

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.

Время выполнения “чистой” программы – 0.139 сек.

Время выполнения программы с библиотеками – 0.351 сек.

Время выполнения программы с регулярными выражениями – 0.102 сек.

# Итоги

Я больше узнал и научился работать с различными языками разметки, написал парсер на языке Python и провёл анализ времени выполнения программ, использующих различные дополнительные библиотеки