МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«Санкт–Петербургский государственный университет  
аэрокосмического приборостроения»

Кафедра №43 «Компьютерных технологий и программной инженерии»

ОТЧЁТ ПО ПРАКТИКЕ

ЗАЩИЩЁН С ОЦЕНКОЙ

Руководитель

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | М.Д. Поляк |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

ОТЧЁТ ПО ПРАКТИКЕ

|  |  |  |
| --- | --- | --- |
| вид практики | производственная | |
| тип практики | научно-исследовательская работа | |
| на тему индивидуального задания | | Модуль проверки отчетов в формате MS Word |
|  | | | |
|  | | | |

|  |  |
| --- | --- |
| выполнен | Цыганковой Екатериной Александровной |
| фамилия, имя, отчество обучающегося в творительном падеже | |

|  |  |  |  |
| --- | --- | --- | --- |
| по направлению подготовки | 02.03.03 |  | Математическое обеспечение и |
|  | код |  | наименование направления |
| администрирование информационных систем | | | |
| наименование направления | | | |
| направленности |  |  | Проектирование программных систем |
|  | код |  | наименование направленности |
|  | | | |
| наименование направленности | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Обучающийся группы № | 4936 |  |  |  | Е.А. Цыганкова |
|  | номер |  | подпись, дата |  | инициалы, фамилия |

Санкт–Петербург 2022

Оглавление

[Цель работы 3](#_Toc108356094)

[Задание 3](#_Toc108356095)

[Входные и выходные данные 3](#_Toc108356096)

[Требования к работе модуля 4](#_Toc108356097)

[Анализ библиотек 5](#_Toc108356098)

[Практический раздел 6](#_Toc108356099)

[Результаты 8](#_Toc108356100)

[Выводы 9](#_Toc108356101)

[Список литературы 10](#_Toc108356102)

[Приложение А 11](#_Toc108356103)

# ****Цель работы****

Реализовать модуль для проверки отчетов студентов, выполненных в формате MS Word (docx).

# ****Задание****

1. Провести анализ существующих библиотек языка Python для парсинга файлов заданного формата. Составить сравнительную таблицу, указав, в том числе: дату выхода последней версии библиотеки; лицензию, по которой распространяется библиотека; наличие внешних зависимостей, которые необходимо устанавливать отдельно; количество звезд на гитхабе, количество форков и открытых Issues; наличие подробной документации; и др.
2. Проанализировать функциональные возможности найденых библиотек для решения поставленной задачи.
3. Согласовать окончательный выбор библиотеки для парсинга файлов.
4. Реализовать поставленную задачу.
5. Подготовить тестовые данные (файлы с отчетами в нужном формате) для проверки корректности работы программы. На каждый проверяемый в отчете элемент должен быть как минимум один отдельный тест, в котором проверяется, что в случае некорректного заполнения соответствующей части отчета, написанная программа генерирует все необходимые сообщения о содержащейся в отчете ошибке. Предусмотреть тесты с различными вариантами оформления отчета (в т.ч. титульного листа). Также тестовые данные должны включать несколько правильно выполненных отчетов, а также неправильно выполненные отчеты, содержащие сразу несколько ошибок.
6. Подготовить тесты с использованием библиотеки pytest.
7. Написать CI/CD процесс с использованием GitHub Actions для автоматического запуска тестов при изменении файла (-ов) в репозитории, в которых реализован функционал данной задачи.

# ****Входные и выходные данные****

Входные данные:

* файл с отчетом в виде потока байт;
* словарь с информацией о студенте;
* словарь с информацией об отчете.

Выходные данные:

Список строк с описанием ошибок в отчете, по одной строке на каждую ошибку.

Пример:

[

"Неправильное название предмета",

"Неверное ФИО студента",

"Неверная дата"

]

*Информация о студенте:*

{

"name": "Иван",

"surname": "Иванов",

"patronymic": "Иванович",

"group": "4931"

}

Поле patronymic может быть пустым (None в Python) или ключ может отсутствовать вовсе. Поле group может содержать буквы, поэтому тип – строка.

*Информация об отчете:*

{

"subject\_name": "Операционные системы",

"task\_name": "ЛР1. Знакомство с командным интерпретатором bash",

"task\_type": "Лабораторная работа",

"teacher": {

"name": "Юлия",

"surname": "Антохина",

"patronymic": "Анатольевна",

"status": "Ректор, д.т.н., проф."

},

"report\_structure": [

"Цель", "Задание", "Результат выполнения", "Выводы"

],

"uploaded\_at": "2022-06-01T00:00:00Z"

}

Поле uploaded\_at содержит дату загрузки студентом отчета в личный кабинет в формате ISO 8601 в часовом поясе UTC.

# Требования к работе модуля

Для решения поставленной задачи необходимо написать функцию на Python, принимающую на вход все необходимые данные. Файл с кодом должен позволять как использование написанной функции в другом модуле с помощью команды import, так и самостоятельное выполнение кода как отдельного приложения с помощью команды python <имя\_файла>.py <аргументы>.

В отчете необходимо проверять:

* титульный лист на соответствие заданным во входных данных параметрам:
  + тип задания (лабораторная работа, курсовой проект и т.п.),
  + ФИО студента,
  + группа студента,
  + название предмета,
  + название задания (лабораторной работы),
  + ФИО преподавателя,
  + должность преподавателя,
  + год выполнения отчета внизу титульного листа (проверять по дате загрузки отчета)
* содержимое отчета на наличие обязательных разделов, если таковые были указаны.

Программа должна поддерживать произвольный шаблон титульного листа, как скачанный с сайта ГУАП, так и упрощенный, а также любые их вариации, в т.ч. с любым оформлением (размер шрифта; выделение слов жирным, курсивом и т.п.; использование невидимых таблиц для форматирования текста, использование разрывов страниц и пр.).

# ****Анализ библиотек****

Таблица 1. Сравнительную таблицу библиотек языка Python для парсинга файлов формата MS Word.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Название библиотеки** | **Дата выхода последней версии** | **Лицензия** | **Наличие внешних зависимостей** | **Кол-во звезд на гитхабе, кол-во форков и открытых Issues** | **Наличие подробной документации** |
| **python-docx** | **16 мая 2021** | **MIT License** | **lxml** | **3200, 869, 442** | **Присутствует** |
| python-docx2txt | **23 июня 2019** | **MIT License** | **-** | **361, 77, 15** | **Отсутствует** |
| **docx2python** | **30 декабря 2021** | **MIT License** | **lxml** | **54, 21, 3** | **Присутствует** |

*MIT License* – короткая и простая разрешительная лицензия с условиями, требующими только сохранения авторских прав и уведомлений о лицензиях. Лицензионные работы, модификации и более крупные работы могут распространяться на разных условиях и без исходного кода.

Проанализировав функциональные возможности найденных библиотек для решения поставленной задачи, было принято решение использовать библиотеку python-docx2txt, т.к. данный модуль может извлекать текст из таблиц, что полезно при работе с файлами, в которых невидимые таблицы использованы для форматирования текста титульного листа.

# ****Практический раздел****

Для проверки содержимого отчета была реализована функция:

def check\_docx(infstudent, infreport, document) **–** которая принимает словарь с информацией о студенте, словарь с информацией об отчете и файл отчета в виде буферизированного потока байт.

С помощью библиотеки python-docx2txt был извлечен текст файла и записан в список по абзацам:

result = docx2txt.process(document)

#добавление в список непустых строк, извлеченных из документа

content = []

for line in result.splitlines():

if line != '':

content.append(line)

При поиске первого вхождения ключевых слов в отчете, если таковые были найдены, в список записывается индекс списка content, а также номер позиции в строке (по умолчанию -1;-1). Например:

struc\_info = [ #наименование (или индекс массива), номер строки, номер позиции

["Должность преподавателя", -1, -1],

["ФИО преподавателя", 7, 0],

["Тип задания", 8, 10],

["Название работы", -1, -1],

["Название предмета",10, 20],

["Номер группы", 12, 0],

["ФИО студента", 13, 0],

["Год",15, 0]

]

Также в список добавляется информация о структуре отчета:

for i in range(len(infreport["report\_structure"])):

struc\_info.append([i,-1,-1])

Далее из списка удаляются все элементы, что не были найдены, и список сортируется, сравнивается с верной структурой всего отчета. Если порядок был нарушен, генерируется ошибка, о том какие элементы отчета нарушили его структуру.

#удалить все ненайденные элементы (-1)

struc\_info1 = [s for s in struc\_info if s[1] != -1]

#если список непустой

if len(struc\_info1) != 0:

#сортируем список по строкам и сравниваем с верной структурой отчета

#выводим, что нарушает последовательность и верную последовательность с первого элемента, нарушающего ее

struc\_info\_correct\_order = list(struc\_info1)

struc\_info1.sort(key=itemgetter(1, 2))

lst = []

lst\_cor = []

k = -1

for i in range(len(struc\_info\_correct\_order)):

if struc\_info1[i][0] != struc\_info\_correct\_order[i][0]:

if k == -1:

k = struc\_info\_correct\_order[i][0]

if type(struc\_info\_correct\_order[i][0]) == int:

lst.append('"' + infreport["report\_structure"][struc\_info\_correct\_order[i][0]] + '"')

else:

lst.append('"'+struc\_info\_correct\_order[i][0]+'"')

**Написание тестов с использованием библиотеки pytest**

Для проверки корректного выполнения работы функции было написано 27 тестов:

10 тестов для проверки работы функции при передаче неправильных входных данных;

6 тестов для правильных отчетов (лабораторная работа, контрольная работа, реферат, курсовая работа, курсовой проект, лабораторная работа с упрощенным оформлением титульного листа);

9 тестов для каждого проверяемого в отчете элемента;

1 тест для проверки правильной структуры отчета;

1 тест для отчета с несколькими ошибками.

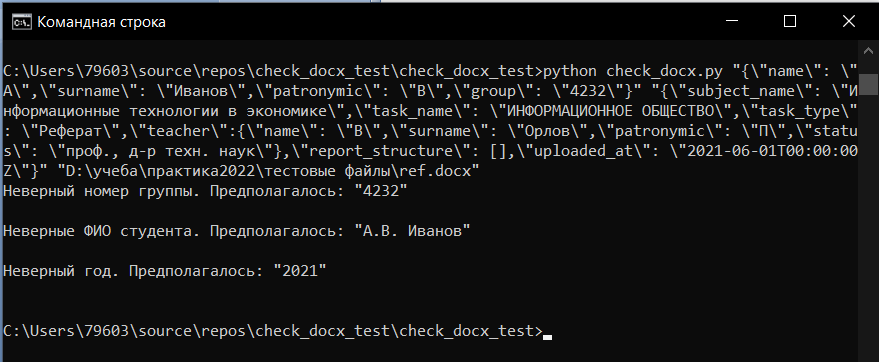
Полный код модулей приведен в приложении А.

# ****Результаты****

Результатом работы является консольное приложение, которое может быть запущено с помощью команды - python check\_docx.py <аргументы>. Например, в командной строке windows запустить приложение можно так:

python check\_docx.py "{\"name\": \"А\", \"surname\": \"Иванов\", \"patronymic\": \"В\", \"group\": \"4232\"}" "{\"subject\_name\": \"Информационные технологии в экономике\", \"task\_name\": \"ИНФОРМАЦИОННОЕ ОБЩЕСТВО\", \"task\_type\": \"Реферат\", \"teacher\": {\"name\": \"В\", \"surname\": \"Орлов\", \"patronymic\": \"П\", \"status\": \"проф., д-р техн. наук\"}, \"report\_structure\": [], \"uploaded\_at\": \"2021-06-01T00:00:00Z\"}" "D:\учеба\практика2022\тестовые файлы\ref.docx"

Скриншот командной строки:



# ****Выводы****

В результате выполнения индивидуального задания был реализован модуль для проверки отчетов студентов, выполненных в формате MS Word (docx).

К достоинствам алгоритма проверки следует отнести его простоту реализации и возможность проверять отчеты произвольной структуры. Недостаток алгоритма в том, что он ищет только первое вхождение ключевого слова и проверяет, находится ли оно на корректной позиции относительно других найденных слов.

Таким образом, можно заключить, что выполненная работа соответствует поставленной задаче.

Ссылка на репозиторий с кодом: <https://github.com/markpolyak/pro.guap-report-validator.git> (ветка - validator-docx)

# ****Список литературы****

1. **How to read Word documents with Python [Электронный ресурс] –** TheAutomatic**, 2019 – URL:** <http://theautomatic.net/2019/10/14/how-to-read-word-documents-with-python/> **(дата обращения: 28.07.2022)**
2. **Репозиторий на GitHub утилиты python-docx2txt [Электронный ресурс] – GitHub, 2019 – URL:** <https://github.com/ankushshah89/python-docx2txt> **(дата обращения: 29.07.2022)**
3. **Класс io.BytesIO() модуля io в Python [Электронный ресурс] –DOCS-Python.ru , 2022 – URL:** <https://docs-python.ru/standart-library/modul-io-python/klass-io-bytesio-modulja-io/> **(дата обращения: 29.07.2022)**
4. **Документация: pytest [Электронный ресурс] –** URL: <https://pytest-docs-ru.readthedocs.io/_/downloads/ru/latest/pdf/> **(дата обращения: 04.07.2022)**
5. **Building and testing Python [Электронный ресурс] – GitHub, 2022 –** URL: <https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python> **(дата обращения: 05.07.2022)**

# Приложение А

*Листинг check\_docx.py*

import io

import json

import docx2txt

import datetime

import sys

from operator import itemgetter

def check\_docx(infstudent, infreport, document) -> list:

if type(infstudent) != dict:

raise TypeError('Информация о студенте не является словарем')

if type(infreport) != dict:

raise TypeError('Информация об отчете не является словарем')

if type(infreport["teacher"]) != dict:

raise TypeError('Ключ "teacher" словаря отчета не является словарем')

#проверка на количество ключей в словаре отчета

if len(infreport.keys()) != 6:

raise ValueError('В словаре отчета недостаточное количество ключей')

#проверка на количество ключей в словаре студента

len\_stud = len(infstudent.keys())

if not (len\_stud == 4 or (len\_stud == 3 and 'patronymic' not in infstudent.keys())):

raise ValueError('В словаре студента недостаточное количество ключей')

#проверка на количество ключей в словаре преподавателя

if type(infreport["teacher"]) == dict:

len\_teach = len(infreport["teacher"].keys())

if not (len\_teach == 4 or (len\_teach == 3 and 'patronymic' not in infreport["teacher"].keys())):

raise ValueError('В словаре учителя недостаточное количество ключей')

#проверка на пустую строку ключей словаря студента

for k in infstudent.keys():

if infstudent[k] == '':

raise ValueError('В словаре студента ключ "'+k+'" является пустой строкой')

#проверка на пустую строку ключей словаря отчета

for k in infreport.keys():

if infreport[k] == '':

raise ValueError('В словаре отчета ключ "'+k+'" является пустой строкой')

#проверка на пустую строку ключей словаря преподавателя

for k in infreport["teacher"].keys():

if infreport["teacher"][k] == '':

raise ValueError('В словаре учителя ключ "'+k+'" является пустой строкой')

#извлечение содержимого документа

result = docx2txt.process(document)

#строка фио студента

stud\_fio = infstudent["name"][0] + "."

if "patronymic" in infstudent.keys() and infstudent["patronymic"] != None:

stud\_fio += infstudent["patronymic"][0] + "."

stud\_fio+= ' '+ infstudent["surname"]

#строка фио преподавателя

teach\_fio = infreport["teacher"]["name"][0] + "."

if "patronymic" in infreport["teacher"].keys() and infreport["teacher"]["patronymic"] != None:

teach\_fio += infreport["teacher"]["patronymic"][0] + "."

teach\_fio+= ' '+ infreport["teacher"]["surname"]

#дата загрузки отчета

uploaded\_at = datetime.datetime.strptime(infreport["uploaded\_at"], '%Y-%m-%dT%H:%M:%SZ')

#добавление в список непустых строк, извлеченнных из документа

content = []

for line in result.splitlines():

if line != '':

content.append(line)

#список с ошибками

errors = list()

#список структуры отчета с найденными и не найденными элементами

struc\_info = [ #наименование (или индекс массива), номер строки, номер позиции

["Должность преподавателя", -1, -1],

["ФИО преподавателя", -1, -1],

["Тип задания", -1, -1],

["Название работы", -1, -1],

["Название предмета",-1, -1],

["Номер группы", -1, -1],

["ФИО студента", -1, -1],

["Год",-1, -1]

]

N = len(content)

#если указаны обязательные разделы отчета

if "report\_structure" in infreport.keys() and infreport["report\_structure"] !=None and len(infreport["report\_structure"])!=0:

#добавляем в список индексы массива

for i in range(len(infreport["report\_structure"])):

struc\_info.append([i,-1,-1])

#ищем ключевые слова обязательных разделов отчета

for s in range(len(infreport["report\_structure"])):

for line in range(N):

struc\_info[8+s][2] = content[line].find(infreport["report\_structure"][s])

if struc\_info[8+s][2] != -1:

struc\_info[8+s][1] = line

break

#тип задания

if "лабораторная работа" in infreport["task\_type"].lower():

for line in range(N):

struc\_info[2][2] = content[line].find(infreport["task\_type"])

if struc\_info[2][2] != -1:

struc\_info[2][1] = line

break

else:

struc\_info[2][2] = content[line].find("ЛАБОРАТОРНОЙ РАБОТЕ")

if struc\_info[2][2] !=-1:

struc\_info[2][1] = line

break

else:

struc\_info[2][2] = content[line].find("лабораторной работе")

if struc\_info[2][2] !=-1:

struc\_info[2][1] = line

break

else:

struc\_info[2][2] = content[line].find("Лабораторная работа")

if struc\_info[2][2] !=-1:

struc\_info[2][1] = line

break

elif "контрольная работа" in infreport["task\_type"].lower():

for line in range(N):

struc\_info[2][2] = content[line].find(infreport["task\_type"])

if struc\_info[2][2] != -1:

struc\_info[2][1] = line

break

else:

struc\_info[2][2] = content[line].find("КОНТРОЛЬНАЯ РАБОТА")

if struc\_info[2][2] != -1:

struc\_info[2][1] = line

break

else:

struc\_info[2][2] = content[line].find("Контрольная работа")

if struc\_info[2][2] != -1:

struc\_info[2][1] = line

break

elif "реферат" in infreport["task\_type"].lower():

for line in range(N):

struc\_info[2][2] = content[line].find(infreport["task\_type"])

if struc\_info[2][2] != -1:

struc\_info[2][1] = line

break

else:

struc\_info[2][2] = content[line].find("РЕФЕРАТ")

if struc\_info[2][2] == 0:

struc\_info[2][1] = line

break

else:

struc\_info[2][2] = content[line].find("Реферат")

if struc\_info[2][2] == 0:

struc\_info[2][1] = line

break

elif "курсовая работа" in infreport["task\_type"].lower():

for line in range(N):

struc\_info[2][2] = content[line].find(infreport["task\_type"])

if struc\_info[2][2] != -1:

struc\_info[2][1] = line

break

else:

struc\_info[2][2] = content[line].find("КУРСОВОЙ РАБОТЕ")

if struc\_info[2][2] != -1:

struc\_info[2][1] = line

break

else:

struc\_info[2][2] = content[line].find("курсовой работе")

if struc\_info[2][2] != -1:

struc\_info[2][1] = line

break

elif "курсовой проект" in infreport["task\_type"].lower():

for line in range(N):

struc\_info[2][2] = content[line].find(infreport["task\_type"])

if struc\_info[2][2] != -1:

struc\_info[2][1] = line

break

else:

struc\_info[2][2] = content[line].find("КУРСОВОМУ ПРОЕКТУ")

if struc\_info[2][2] != -1:

struc\_info[2][1] = line

break

else:

struc\_info[2][2] = content[line].find("курсовому проекту")

if struc\_info[2][2] != -1:

struc\_info[2][1] = line

break

for line in range(N):

#должность преподавателя

struc\_info[0][2] = content[line].find(infreport["teacher"]["status"])

if struc\_info[0][2] != -1 and struc\_info[0][1] == -1:

struc\_info[0][1] = line

#ФИО преподавателя

struc\_info[1][2] = content[line].find(teach\_fio)

if struc\_info[1][2] != -1 and struc\_info[1][1] == -1:

struc\_info[1][1] = line

#название работы

struc\_info[3][2] = content[line].find(infreport["task\_name"])

if struc\_info[3][2] != -1 and struc\_info[3][1] == -1:

struc\_info[3][1] = line

else:

struc\_info[3][2] = content[line].find(infreport["task\_name"].upper())

if struc\_info[3][2] != -1 and struc\_info[3][1] == -1:

struc\_info[3][1] = line

#название предмета

struc\_info[4][2] = content[line].find(infreport["subject\_name"])

if struc\_info[4][2] != -1 and struc\_info[4][1] == -1:

struc\_info[4][1] = line

else:

struc\_info[4][2] = content[line].find(infreport["subject\_name"].upper())

if struc\_info[4][2] != -1 and struc\_info[4][1] == -1:

struc\_info[4][1] = line

else:

struc\_info[4][2] = content[line].find(infreport["subject\_name"].lower())

if struc\_info[4][2] != -1 and struc\_info[4][1] == -1:

struc\_info[4][1] = line

#номер группы

struc\_info[5][2] = content[line].find(str(infstudent["group"]))

if struc\_info[5][2] != -1 and struc\_info[5][1] == -1:

struc\_info[5][1] = line

#ФИО студента

struc\_info[6][2] = content[line].find(stud\_fio)

if struc\_info[6][2] != -1 and struc\_info[6][1] == -1:

struc\_info[6][1] = line

#год

struc\_info[7][2] = content[line].find(str(uploaded\_at.year))

if struc\_info[7][2] != -1 and struc\_info[7][1] == -1:

struc\_info[7][1] = line

#если что-то не найдено в титульнике

if struc\_info[0][1] == -1:

errors.append('Неверная должность преподавателя. Предполагалось: "'+infreport["teacher"]["status"]+'"')

if struc\_info[1][1] == -1:

errors.append('Неверные ФИО преподавателя. Предполагалось: "'+ teach\_fio+'"')

if struc\_info[2][1] == -1:

errors.append('Неверный тип задания. Предполагалось: "'+ infreport["task\_type"]+'"')

if struc\_info[3][1] == -1:

errors.append('Неверное название работы. Предполагалось: "'+ infreport["task\_name"]+'"')

if struc\_info[4][1] == -1:

errors.append('Неверное название предмета. Предполагалось: "'+ infreport["subject\_name"]+'"')

if struc\_info[5][1] == -1:

errors.append('Неверный номер группы. Предполагалось: "'+ str(infstudent["group"])+'"')

if struc\_info[6][1] == -1:

errors.append('Неверные ФИО студента. Предполагалось: "'+ stud\_fio+'"')

if struc\_info[7][1] == -1:

errors.append('Неверный год. Предполагалось: "'+ str(uploaded\_at.year)+'"')

if len(struc\_info) > 8:

#если что-то не найдено в отчете

for i in range(8, len(struc\_info)):

if struc\_info[i][1] == -1:

errors.append('Неверная структура отчета. Не найдено: "'+ infreport["report\_structure"][struc\_info[i][0]]+'"')

#структура отчета с найденными и ненайденными элементами

struc\_info\_cor = list(struc\_info)

#далее работаем только с теми, что были найдены

#удалить все ненайденные элементы (-1)

struc\_info1 = [s for s in struc\_info if s[1] != -1]

#если список непустой, что-то было найдено

if len(struc\_info1) != 0:

#сортируем список по строкам и сравниваем с верной структурой отчета

#выводим что нарушает последовательность и верную последовательсноть с первого элемента, нарушающего ее

struc\_info\_correct\_order = list(struc\_info1)

struc\_info1.sort(key=itemgetter(1, 2))

lst = []

lst\_cor = []

k = -1

for i in range(len(struc\_info\_correct\_order)):

if struc\_info1[i][0] != struc\_info\_correct\_order[i][0]:

if k == -1:

k = struc\_info\_correct\_order[i][0]

if type(struc\_info\_correct\_order[i][0]) == int:

lst.append('"'+infreport["report\_structure"][struc\_info\_correct\_order[i][0]]+'"')

else:

lst.append('"'+struc\_info\_correct\_order[i][0]+'"')

#составляем список верной структуры (даже с теми элементами, которые были пропущены)

if len(lst) != 0:

for l in range(len(struc\_info\_cor)):

if struc\_info\_cor[l][0] == k:

for i in range(l,len(struc\_info\_cor)):

if type(struc\_info\_cor [i][0]) == int:

lst\_cor.append('"'+infreport["report\_structure"][struc\_info\_cor[i][0]]+'"')

else:

lst\_cor.append('"'+struc\_info\_cor[i][0]+'"')

break

delim = ', '

errors.append("Элементы отчета: "+delim.join(lst) + " — нарушают его корректную структуру. Верная структура: "+delim.join(lst\_cor)+'.')

return errors

if \_\_name\_\_ == "\_\_main\_\_":

# вызов из консоли

if len(sys.argv) == 4:

#преобразуем в словари принятые аргументы

arg1 = json.loads(sys.argv[1])

arg2 = json.loads(sys.argv[2])

#file = sys.argv[3] #путь к файлу

#передаем в функцию буферизированный поток байт

#check\_docx(arg1, arg2, io.BytesIO(open("{}".format(file),'rb').read()))

file = open("{}".format(sys.argv[3]),'rb')

res = check\_docx(arg1, arg2, io.BytesIO(file.read()))

file.close()

for i in res:

print(i+'\n')

elif len(sys.argv) > 4:

print("Слишком много аргументов")

elif len(sys.argv) < 4:

print("Слишком мало аргументов")

*Листинг check\_docx\_test.py*

from check\_docx import check\_docx

import io

import pytest

from copy import deepcopy

student = {

"name": "Екатерина",

"surname": "Цыганкова",

"patronymic": "Александровна",

"group": "4936"

}

report = {

"subject\_name": "Операционные системы",

"task\_name": "ЛР2. Работа с текстовыми потоками в командном интерпретаторе Bash",

"task\_type": "Лабораторная работа",

"teacher": {

"name": "Марк",

"surname": "Поляк",

"patronymic": "Дмитриевич",

"status": "Старший преподаватель"

},

"report\_structure": [

"Цель","Индивидуальное задание", "Описание входных данных", "Результат выполнения работы", "Исходный код программы с комментариями", "Выводы"

],

"uploaded\_at": "2022-06-01T00:00:00Z"

}

f1 = open('docx\_files//lab.docx','rb')

f2 = open('docx\_files//lab\_title.docx','rb')

f3 = open('docx\_files//lab\_wrong\_struc.docx','rb')

f4 = open('docx\_files//kontr.docx','rb')

f5 = open('docx\_files//ref.docx','rb')

f6 = open('docx\_files//kursrab.docx','rb')

f7 = open('docx\_files//kurspr.docx','rb')

f8 = open('docx\_files//lab\_lot\_err.docx','rb')

reports = {

"lab": f1.read(),

"lab\_title" : f2.read(),

"lab\_wrong\_struc" : f3.read(),

"kontr" : f4.read(),

"ref" : f5.read(),

"kursrab" : f6.read(),

"kurspr" : f7.read(),

"lab\_lot\_err" : f8.read()

}

"""

reports = {

"lab": open('docx\_files//lab.docx','rb').read(),

"lab\_title" : open('docx\_files//lab\_title.docx','rb').read(),

"lab\_wrong\_struc" : open('docx\_files//lab\_wrong\_struc.docx','rb').read(),

"kontr" : open('docx\_files//kontr.docx','rb').read(),

"ref" : open('docx\_files//ref.docx','rb').read(),

"kursrab" : open('docx\_files//kursrab.docx','rb').read(),

"kurspr" : open('docx\_files//kurspr.docx','rb').read(),

"lab\_lot\_err" : open('docx\_files//lab\_lot\_err.docx','rb').read()

}

"""

#тесты для проверки входных данных

#словарь отчета без одного ключа

def test\_rep\_without\_key():

rep = deepcopy(report)

rep.pop("uploaded\_at")

with pytest.raises(ValueError, match=r"В словаре отчета недостаточное количество ключей") as e:

check\_docx(student, rep, io.BytesIO(reports["lab"]))

#assert e.type == ValueError

#assert e.value == ValueError('В словаре отчета недостаточное количество ключей')

#словарь студента без одного ключа

def test\_stud\_without\_key():

stud = deepcopy(student)

stud.pop("surname")

with pytest.raises(ValueError, match=r"В словаре студента недостаточное количество ключей") as e:

check\_docx(stud, report, io.BytesIO(reports["lab"]))

#словарь студента без ключа отчества

#возвращается не код ошибки, программа завершается корректно с

#выводом ошибки о неверных фио студента

def test\_stud\_without\_patr():

stud = deepcopy(student)

stud.pop("patronymic")

res = check\_docx(stud, report, io.BytesIO(reports["lab"]))

assert len(res) == 1

assert res[0][0:21] == "Неверные ФИО студента"

#словарь студента с none в отчестве

#возвращается не код ошибки, программа завершается корректно с

#выводом ошибки о неверных фио студента

def test\_stud\_none\_patr():

stud = deepcopy(student)

stud["patronymic"] = None

res = check\_docx(stud, report, io.BytesIO(reports["lab"]))

assert len(res) == 1

assert res[0][0:21] == "Неверные ФИО студента"

#словарь учителя без одного ключа

def test\_teach\_without\_key():

rep = deepcopy(report)

rep["teacher"].pop("name")

with pytest.raises(ValueError, match=r"В словаре учителя недостаточное количество ключей") as e:

check\_docx(student, rep, io.BytesIO(reports["lab"]))

#словарь отчета без ключа отчества

#возвращается не код ошибки, программа завершается корректно с

#выводом ошибки о неверных фио преподавателя

def test\_teach\_without\_patr():

rep = deepcopy(report)

rep["teacher"].pop("patronymic")

res = check\_docx(student, rep, io.BytesIO(reports["lab"]))

assert len(res) == 1

assert res[0][0:26] == "Неверные ФИО преподавателя"

#словарь учетиля с None в отчестве

#возвращается не код ошибки, программа завершается корректно с

#выводом ошибки о неверных фио преподавателя

def test\_teach\_none\_patr():

rep = deepcopy(report)

rep["teacher"]["patronymic"] = None

res = check\_docx(student, rep, io.BytesIO(reports["lab"]))

assert len(res) == 1

assert res[0][0:26] == "Неверные ФИО преподавателя"

#словарь студента с пустой строкой в ключе

def test\_stud\_empty\_key():

stud = deepcopy(student)

stud["name"] =''

with pytest.raises(ValueError, match=r"В словаре студента ключ .\* является пустой строкой") as e:

check\_docx(stud, report, io.BytesIO(reports["lab"]))

#словарь отчета с пустой строкой в ключе

def test\_rep\_empty\_key():

rep = deepcopy(report)

rep["task\_name"] = ''

with pytest.raises(ValueError, match=r"В словаре отчета ключ .\* является пустой строкой") as e:

check\_docx(student, rep, io.BytesIO(reports["lab"]))

#словарь учителя с пустой строкой в ключе

def test\_teach\_empty\_key():

rep = deepcopy(report)

rep["teacher"]["patronymic"] = ''

with pytest.raises(ValueError, match=r"В словаре учителя ключ .\* является пустой строкой") as e:

check\_docx(student, rep, io.BytesIO(reports["lab"]))

#тесты для проверки правильных отчетов

#проверка лабораторной работы

def test\_correct\_lab():

res = check\_docx(student, report, io.BytesIO(reports["lab"]))

assert len(res) == 0

#контрольная работа

def test\_correct\_kontr():

rep = {

"subject\_name": "Архитектура ЭВМ",

"task\_name": "ПРЕДСТАВЛЕНИЕ ДАННЫХ В ЭВМ ТИПА VAX-11",

"task\_type": "Контрольная работа",

"teacher": {

"name": "Владимир",

"surname": "Федоров",

"patronymic": "Станиславович",

"status": "доц., канд. техн. наук"

},

"report\_structure": [],

"uploaded\_at": "2022-06-01T00:00:00Z"

}

res = check\_docx(student, rep, io.BytesIO(reports["kontr"]))

assert len(res) == 0

#реферат

def test\_correct\_ref():

rep = {

"subject\_name": "Информационные технологии в экономике",

"task\_name": "ИНФОРМАЦИОННОЕ ОБЩЕСТВО",

"task\_type": "Реферат",

"teacher": {

"name": "Владимир",

"surname": "Орлов",

"patronymic": "Павлович",

"status": "проф., д-р техн. наук"

},

"report\_structure": [],

"uploaded\_at": "2022-06-01T00:00:00Z"

}

res = check\_docx(student, rep, io.BytesIO(reports["ref"]))

assert len(res) == 0

#курсовая работа

def test\_correct\_kursrab():

rep = {

"subject\_name": "Прикладные модели оптимизации",

"task\_name": "Школьные перевозки",

"task\_type": "Курсовая работа",

"teacher": {

"name": "Мария",

"surname": "Фаттахова",

"patronymic": "Владимировна",

"status": "доц., к.ф.-м.н., доцент"

},

"report\_structure": ["Цель","Постановка задачи","Математическая модель","Оптимальное решение задачи","Обоснование выбора языка программирования","Список использованной литературы"],

"uploaded\_at": "2021-06-01T00:00:00Z"

}

res = check\_docx(student, rep, io.BytesIO(reports["kursrab"]))

assert len(res) == 0

#курсовой проект

def test\_correct\_kurspr():

rep = {

"subject\_name": "Структуры и алгоритмы обработки данных",

"task\_name": "Использование заданных структур данных и алгоритмов при разработке программного обеспечения информационной системы",

"task\_type": "Курсовой проект",

"teacher": {

"name": "Валерий",

"surname": "Матьяш",

"patronymic": "Анатольевич",

"status": "доц., канд. техн. наук"

},

"report\_structure": ["Задание","Введение","Алгоритмы и структуры данных","Описание программы","Тестирование программы","Заключение"],

"uploaded\_at": "2021-06-01T00:00:00Z"

}

res = check\_docx(student, rep, io.BytesIO(reports["kurspr"]))

assert len(res) == 0

#лр со свободным титульным листом

def test\_correct\_lab\_title():

res = check\_docx(student, report, io.BytesIO(reports["lab\_title"]))

assert len(res) == 0

#тесты для проверки всех элементов отчета

#проверка структуры отчета

def test\_incorrect\_struture\_title():

res = check\_docx(student, report, io.BytesIO(reports["lab\_wrong\_struc"]))

assert len(res) == 1

assert res[0][0:115] == 'Элементы отчета: "Должность преподавателя", "ФИО преподавателя", "Год", "Цель" — нарушают его корректную структуру.'

#в данном тесте и далее проверяется один отчет, меняются только входные данные

#фио студента

def test\_surname\_student():

stud = deepcopy(student)

stud["surname"] = "Иванов"

res = check\_docx(stud, report, io.BytesIO(reports["lab"]))

assert len(res) == 1

assert res[0][0:21] == "Неверные ФИО студента"

#группа студента

def test\_group\_student():

stud = deepcopy(student)

stud["group"] = 4932

res = check\_docx(stud, report, io.BytesIO(reports["lab"]))

assert len(res) == 1

assert res[0][0:21] == "Неверный номер группы"

#предмет

def test\_subject\_report():

rep = deepcopy(report)

rep["subject\_name"] = "Информационное право"

res = check\_docx(student, rep, io.BytesIO(reports["lab"]))

assert len(res) == 1

assert res[0][0:26] == "Неверное название предмета"

#имя работы

def test\_task\_name\_report():

rep = deepcopy(report)

rep["task\_name"] = "Представление данных в ЭВМ типа VAX-11"

res = check\_docx(student, rep, io.BytesIO(reports["lab"]))

assert len(res) == 1

assert res[0][0:24] == "Неверное название работы"

#тип работы

def test\_task\_type\_report():

rep = deepcopy(report)

rep["task\_type"] = "Реферат"

res = check\_docx(student, rep, io.BytesIO(reports["lab"]))

assert len(res) == 1

assert res[0][0:20] == "Неверный тип задания"

#фио преподавателя

def test\_surname\_teacher():

rep = deepcopy(report)

rep["teacher"]["surname"] = "Антохина"

res = check\_docx(student, rep, io.BytesIO(reports["lab"]))

assert len(res) == 1

assert res[0][0:26] == "Неверные ФИО преподавателя"

#должность преподавателя

def test\_status\_teacher():

rep = deepcopy(report)

rep["teacher"]["status"] = "Ректор, д.т.н., проф."

res = check\_docx(student, rep, io.BytesIO(reports["lab"]))

assert len(res) == 1

assert res[0][0:32] == "Неверная должность преподавателя"

#год

def test\_year\_report():

rep = deepcopy(report)

rep["uploaded\_at"] = "2020-06-01T00:00:00Z"

res = check\_docx(student, rep, io.BytesIO(reports["lab"]))

assert len(res) == 1

assert res[0][0:12] == "Неверный год"

#наличие структуры отчета

def test\_struture\_report():

rep = deepcopy(report)

rep["report\_structure"].append("Окончательные выводы")

res = check\_docx(student, rep, io.BytesIO(reports["lab"]))

assert len(res) == 1

assert res[0][0:61] == "Неверная структура отчета. Не найдено: "+'"Окончательные выводы"'

#отчет по лабораторной, в котором находится

#сразу несколько ошибок

def test\_lab\_lot\_err():

rep = deepcopy(report)

rep["task\_name"] = "ЛР5. Управление памятью"

rep["report\_structure"] = ["Цель","Задание","Описание ипользуемых алгоритмов замещения страниц","Результат выполнения работы","Исходный код программы с комментариями","Выводы"]

res = check\_docx(student, rep, io.BytesIO(reports["lab\_lot\_err"]))

assert len(res) == 5

assert res[0][0:32] == "Неверная должность преподавателя"

assert res[1][0:20] == "Неверный тип задания"

assert res[2][0:12] == "Неверный год"

assert res[3][0:90] == 'Неверная структура отчета. Не найдено: "'+rep["report\_structure"][2]+'"'

assert res[4][0:165] == 'Элементы отчета: "Название работы", "Название предмета", "Результат выполнения работы", "Исходный код программы с комментариями" — нарушают его корректную структуру.'

f1.close()

f2.close()

f3.close()

f4.close()

f5.close()

f6.close()

f7.close()

f8.close()

*Листинг test.yml*

name: Check docx file

on:

push:

paths:

- report-validator/validator\_docx/check\_docx.py

jobs:

test:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Set up Python

uses: actions/setup-python@v4

with:

python-version: '3.8'

- name: Install dependencies

run: |

python -m pip install --upgrade pip setuptools wheel

pip install docx2txt

- name: Test with pytest

run: |

pip install pytest

cd ./report-validator/validator\_docx

pytest check\_docx\_test.py