# Документация модуля для парсинга docx

## Библиотеки

Для начальной работы потребуются следующий список библиотек, представленный на листинге 1.

Листинг 1 — Библиотеки.

import docx

from docx import Document

from docx.text.paragraph import Paragraph

from docx.table import Table

from docx.oxml.text.paragraph import CT\_P as omxl\_paragraph

from docx.oxml.table import CT\_Tbl as omxl\_table

from docx.oxml.section import CT\_SectPr as omxl\_section

import regex

Их можно установить, используя файл requirements.txt, через команду pip install -r requirements.txt. Используются следующие версии библиотек:

* python-docx==1.1.2;
* regex==2024.11.6.

## Открытие файла

Далее, необходимо открыть файл, реализуется с помощью функции *open\_docx,* которая на вход принимает путь к файлу формата docx. Реализация представлена на листинге 2.

Листинг 2 — Открытие файла формата docx.

def open\_docx(path):

return Document(path)

На рисунке 1 показан результат работы листинга 2.

Рисунок 1 — Открытие файла docx

Как можем заметить, что у нас возвращается объект нашего документа, следовательно, он обладает следующими, для нас полезными, свойствами и атрибутами:

* Получение только параграфов, т.е. абзацы, разделенные переносом на другую строку, где каждый параграф есть объект;
* Получение только таблиц, где каждая таблица есть объект;
* Получение всех элементов файла в формате xml.

Стоит более подробно описать каждый из пунктов.

## Получение параграфов

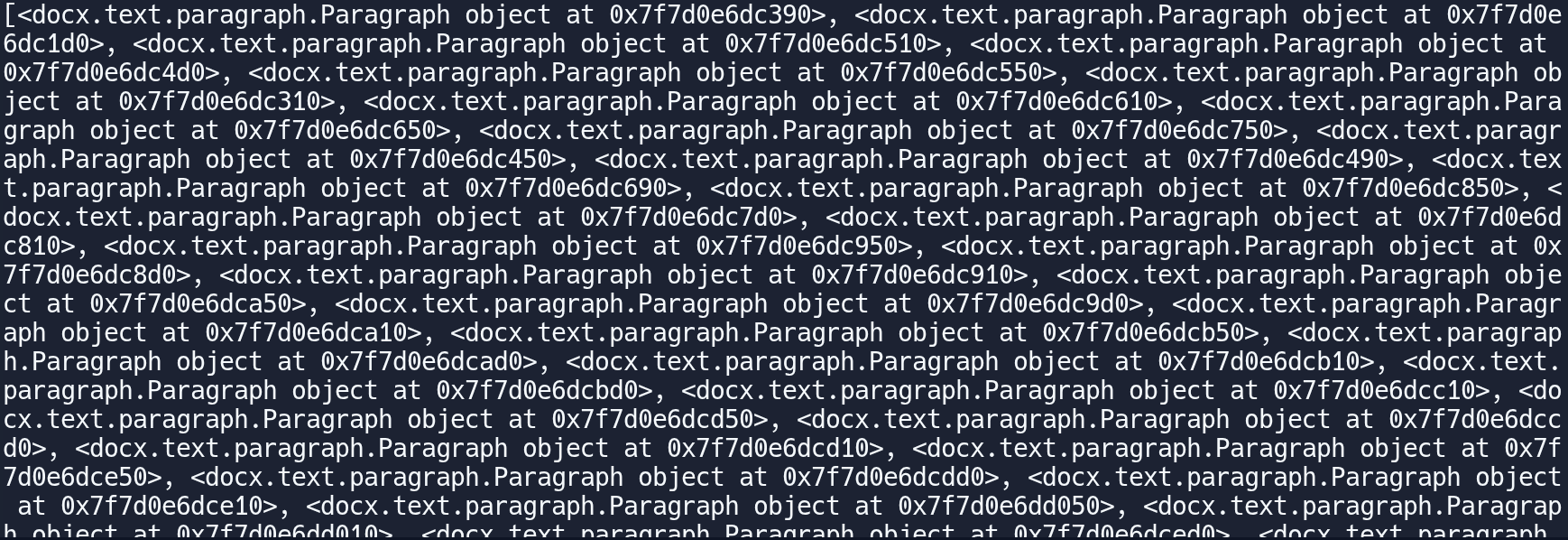
На листинге 3 будет представлено получение только тех элементов, которые являются параграфами.

Листинг 3 — Получение параграфов

def get\_paragraphs(document):

return document.paragraphs

В результате возвращается список объектов *docx.text.paragraph*. Результат представлен на рисунке 2.

Рисунок 2 — Список объектов параграфов

Зная, что параграф другими словами — абзац, а абзац — набор слов, то у MS Word, как и в большинстве текстовых редакторах, присутствуют определенные свойства, такие как:

* Курсив;
* Жирный шрифт;
* Подчеркивание.

Это одни из основных, полный перечень можно узнать в документации MS Word.

В случае поставленной задачи парсинга, необходим сам текст, анализировать свойства не нужно. Получение текста представлено на листинге 4.

Листинг 4 — Получение текста из объектов *docx.text.paragraph*.

def get\_text(paragraph):

return paragraph.text

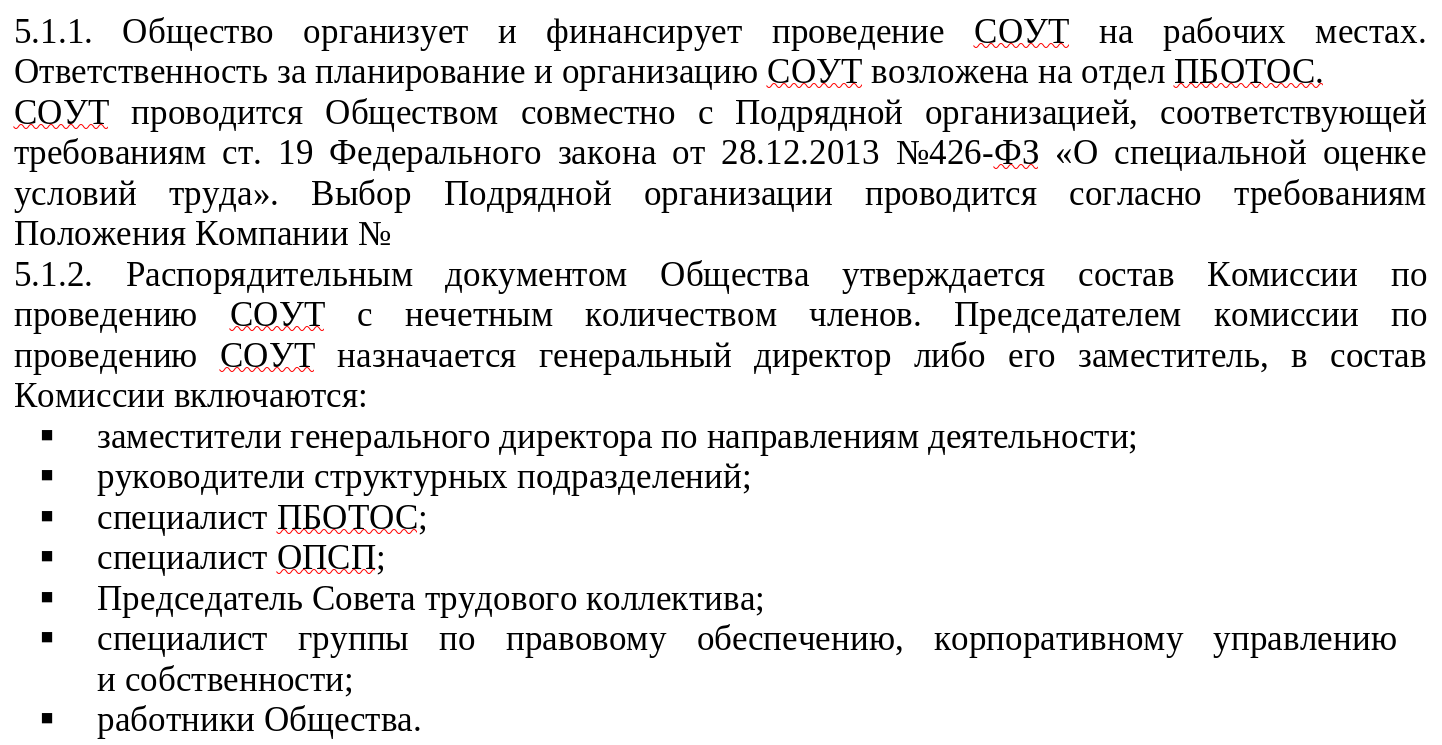
def iter\_paragraphs(paragraphs):

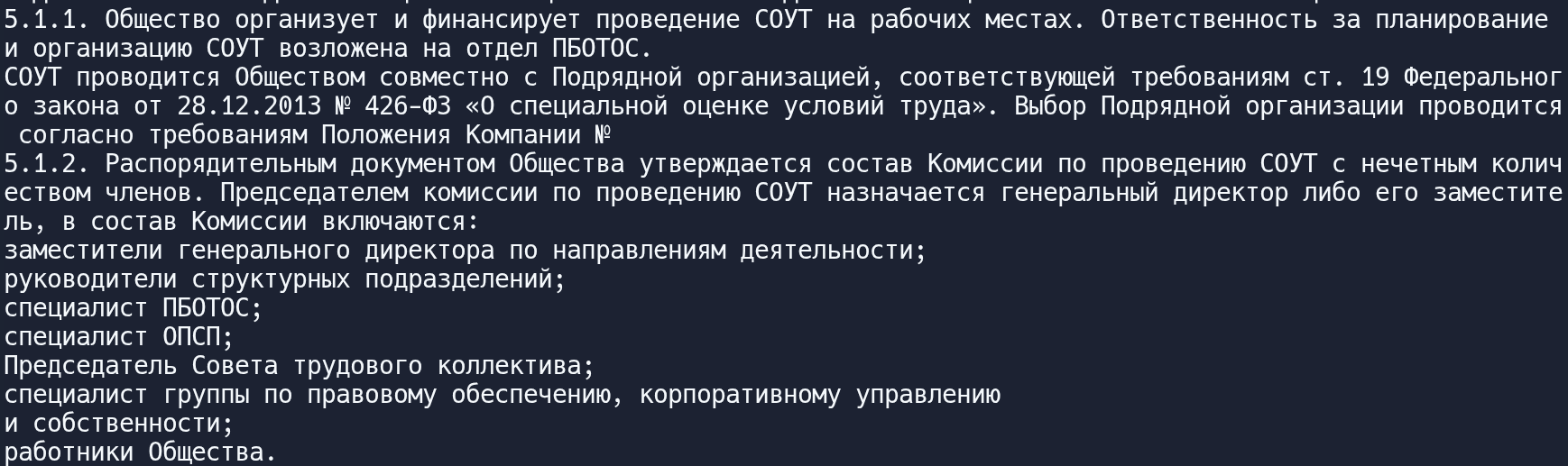
for paragraph in paragraphs:

print(get\_text(paragraph))

iter\_paragraphs(paragraphs)

На рисунке 3 будет представлен текст во входном документе, на рисунке 4 будет представлен результат получение данного текста.

Рисунок 3 — Входной документ

Рисунок 4 — Результат обработанного текста

## Получение таблиц

Одна из специфичных задач — корректная обработка таблиц. Получение таблиц, как и параграфов, не составляет особой сложности. На листинге 5 представлено получение таблиц.

Листинг 5 — Получение списка таблиц.

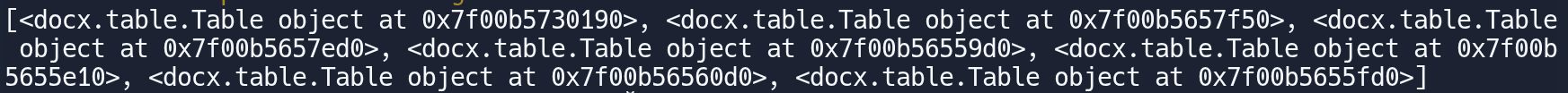
def get\_tables(document):

return document.tables

tables = get\_tables(document)

print(tables)

Тут тоже возвращается список объектов, только уже docx.table. Результат представлен на рисунке 5.

Рисунок 5 — Список объектов таблиц

Как таковых свойств таблиц замечено не было, поэтому остается только представить в читабельном виде. Способ получения представлен на листинге 6.

Листинг 6 — Читабельное представление таблиц.

def iter\_table(table):

for row in table.rows:

\_row = []

for cell in row.cells:

\_row += [cell.text]

print(\_row)

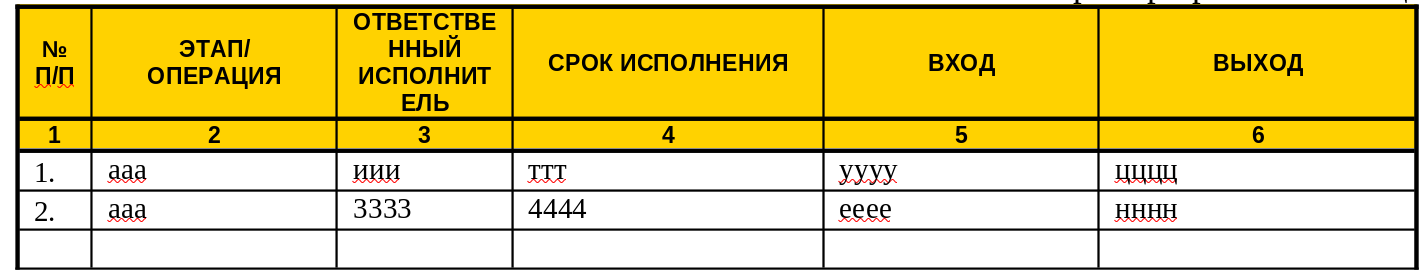
# взяли четвертую

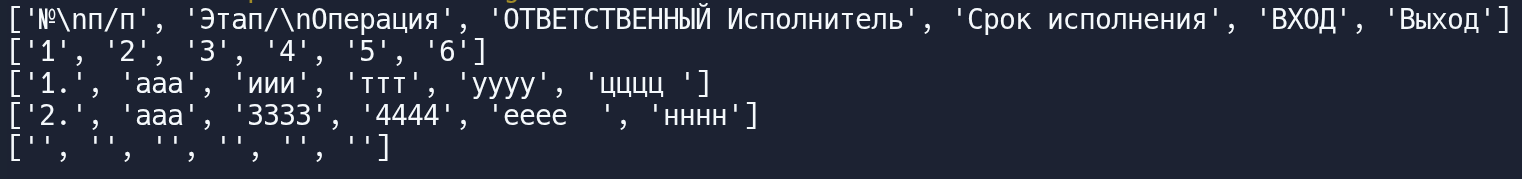
# таблицу из списка

table = tables[3]

iter\_table(table)

Результат работы алгоритма представлен на рисунке 7. На рисунке 6 будет отображена таблица, которая находится во входном файле.

Рисунок 6 — Входная таблица, находящиеся в документе

Рисунок 7 — Результат полученной преобразованной таблицы

Как можно было заметить, что при простой организации таблицы все обрабатывается корректно, но при разделении/объединение колонки/строки, происходит следующий результат, который показан на рисунках 8 — 11.

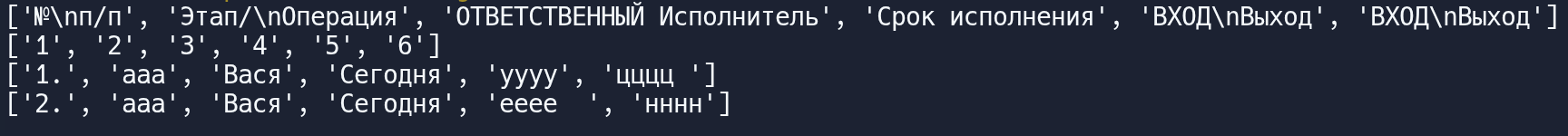
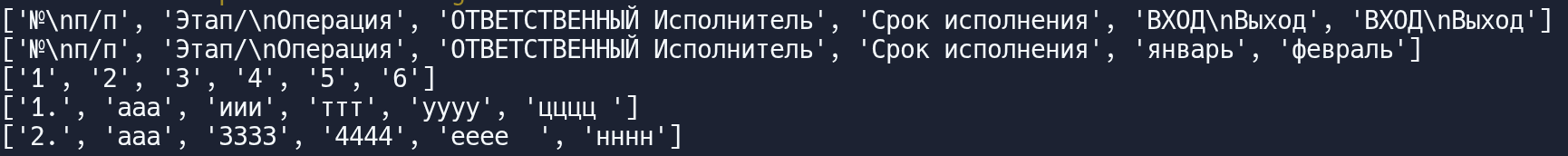
Рисунок 8 — Входная таблица с объединением строк

Рисунок 9 — Результат обработки с объединением строк

Рисунок 10 — Входная таблица с разделением столбцов

Рисунок 11 — Результат обработки с разделением столбцов

Из приведенных выше примеров наблюдается следующая зависимость — при объединении/разделение элементы повторяются, что и логично с точки зрения внутренней структуры xml. Это как раз хорошо повлияет для будущей поставленной задачи. Более детальная обработка таблиц будет показаны в разделе „Обработка данных“.

## Элементы файла типа xml

MS Word по своей структуре представляет собой xml. Если до этого можно было получить отдельно параграфы и таблицы, то через элементы xml, можно последовательно получать каждый из этих элементов. Но не стоит забывать о том, что по мимо параграфов, таблиц в файле могут находиться другие объекты, например, как верхний/нижний колонтитул. Поэтому, необходимо учитывать только текст и таблицы. Данная выборка представлена в листинге 7.

Листинг 7 — Формирование основного тела файла.

def get\_body(document):

body = []

for element in document.element.body:

if isinstance(element, omxl\_paragraph) or isinstance(element, omxl\_table):

body += [element]

return body

body = get\_body(document)

Результатом получим список из объектов xml параграфов и таблиц, как это показано на рисунке 12.

Рисунок 12 — Тело файла

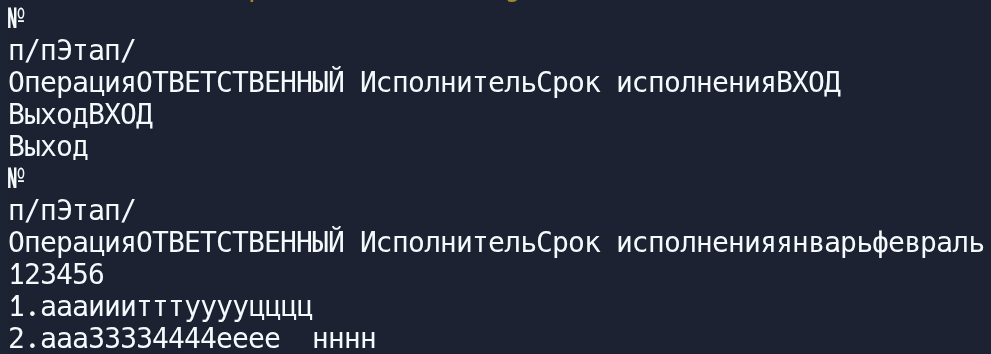
## Обработка данных

После того, как получи все необходимые данные их необходимо между собой объединить. Под этим подразумевается следующие:

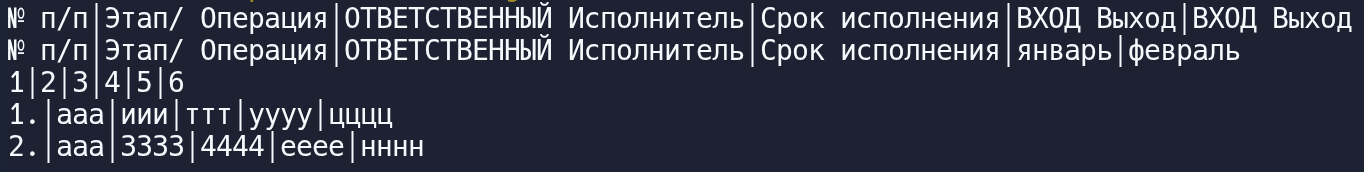
* Преобразовать каждый объект в текст;
* Присоединить к таблице название таблицы.

### Получение всего текста

Обработать объекты параграфа, как и было сказано в разделе „Получение таблиц“, не составляет труда, куда больше идей потребует преобразовать таблицу в читабельный вид. На рисунках 7, 9, 11 пока преобразовали только в строки списков и если же захотим их соединить, то получим следующее (рисунок 13).

Рисунок 13 — Соединение элементов таблицы

Из изображения видно, что какие-то строки и столбцы съехали, да и не понятно, как они между собой связаны, поэтому необходимо сделать определенную *чистку* данных, т.е. удалить лишние отступы, пробелы и т.п. и добавить некий разделитель, как это отображено на рисунке 14.

Рисунок 14 — Чистка данных для таблицы

Так на много лучше, но остается до сих пор не читабельно из-за разной ширины. Следовательно, для каждого столбца необходимо найти элемент, который имеет максимальную длину и выровнять по центу. Алгоритмы представлен в листинге 8 — 9.

Листинг 8 — Поиск максимального по ширине элемента столбца.

def lenght\_every\_column(table, rows, columns):

dict\_lenght\_columns = {}

for i in range(columns):

maxim\_lenght = 0

for j in range(rows):

maxim\_lenght = max(maxim\_lenght, len(table[j][i]))

dict\_lenght\_columns[f'столбец\_{i}'] = maxim\_lenght + 9 if maxim\_lenght % 2 else maxim\_lenght + 8

return dict\_lenght\_columns

Листинг 9 — Форматирование столбцов, учитывая ширину.

def format\_columns(table, rows, columns):

dict\_lenght\_columns = lenght\_every\_column(table, rows, columns)

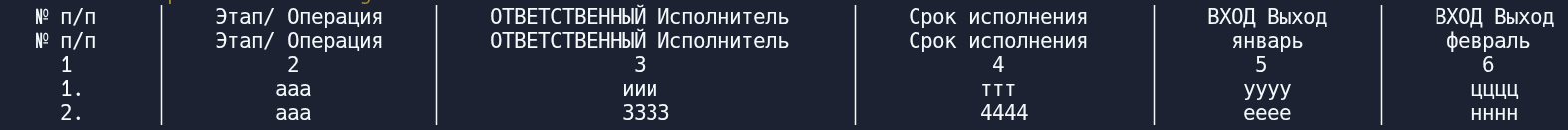
for i in range(columns):

format\_space = dict\_lenght\_columns[f'столбец\_{i}']

for j in range(rows):

table[j][i] = f'{table[j][i]:^{format\_space}}'

Получаем следующий вид таблицы (рисунок 15).

Рисунок 15 — Читабельный вид таблицы

Теперь, когда основная работа выполнена, немного добавим стилистики — разделим на ячейки. Реализация представлена в листинге 10.

Листинг 10 – Разделитель строк на ячейки.

def sep\_rows(row):

    return f"{'─' \* len(row)}"

В результате получаем следующее (рисунок 16).

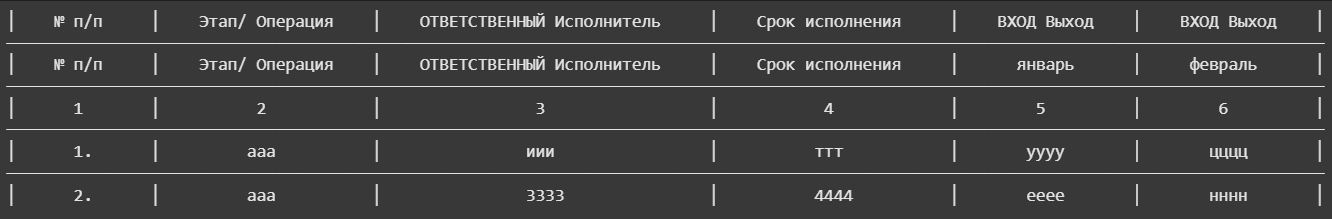


Рисунок 16 — Добавление разделителя

Но, как можно заметить линии между собой не соединены. Зная, про разделяющий символ между элементами, будем находить его индекс и при создании разделителя стоки на данное место будем помещать другой разделяющий символ, это показано в листинге 11.

Листинг 11 — Поиск индексов для замены на разделяющий символ.

def index\_sep\_rows(row, columns):

    index\_sep = []

    count = 0

    while count != columns:

        index = row[1:].index('│')

        index\_sep += [index + 1]

        row = row[index + 1:]

        count += 1

    return index\_sep

Теперь, перейдем к замене, тогда модифицируем листинг 10.

Листинг 10 – Разделитель строк на ячейки (модификация).

def sep\_rows(index\_sep):

    return f"├{'┼'.join(list(map(lambda x: '─' \* (x - 1), index\_sep)))}┤"

Получаем следующее (рисунок 17).

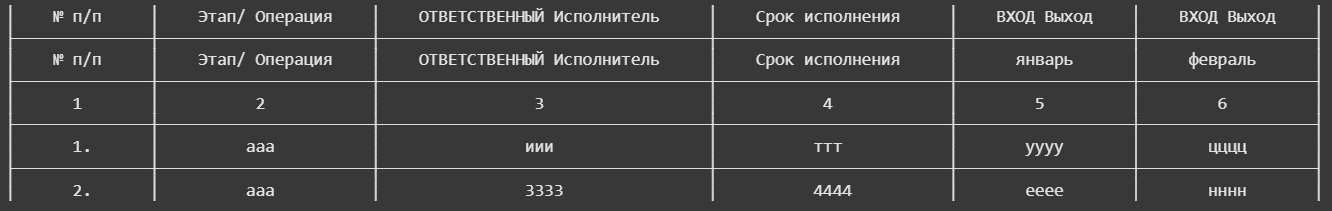


Рисунок 17 — Модификация разделителя

Остается добавить верхний и нижний разделитель, отличаются только в элементах разделения. Они представлены на листингах 12 – 13.

Листинг 12 – Разделитель строки для начальной строки

def sep\_begin\_row(index\_sep):

    return f"├{'┬'.join(list(map(lambda x: '─' \* (x - 1), index\_sep)))}┤"

Листинг 13 – Разделитель строки для последней строки

def sep\_last\_row(index\_sep):

    return f"└{'┴'.join(list(map(lambda x: '─' \* (x - 1), index\_sep)))}┘"

В результате получаем почти полноценную таблицу (рисунок 18).



Рисунок 18 — Полноценная читабельная таблица

Соединим текст и таблицы, полученный результат на рисунке 19.

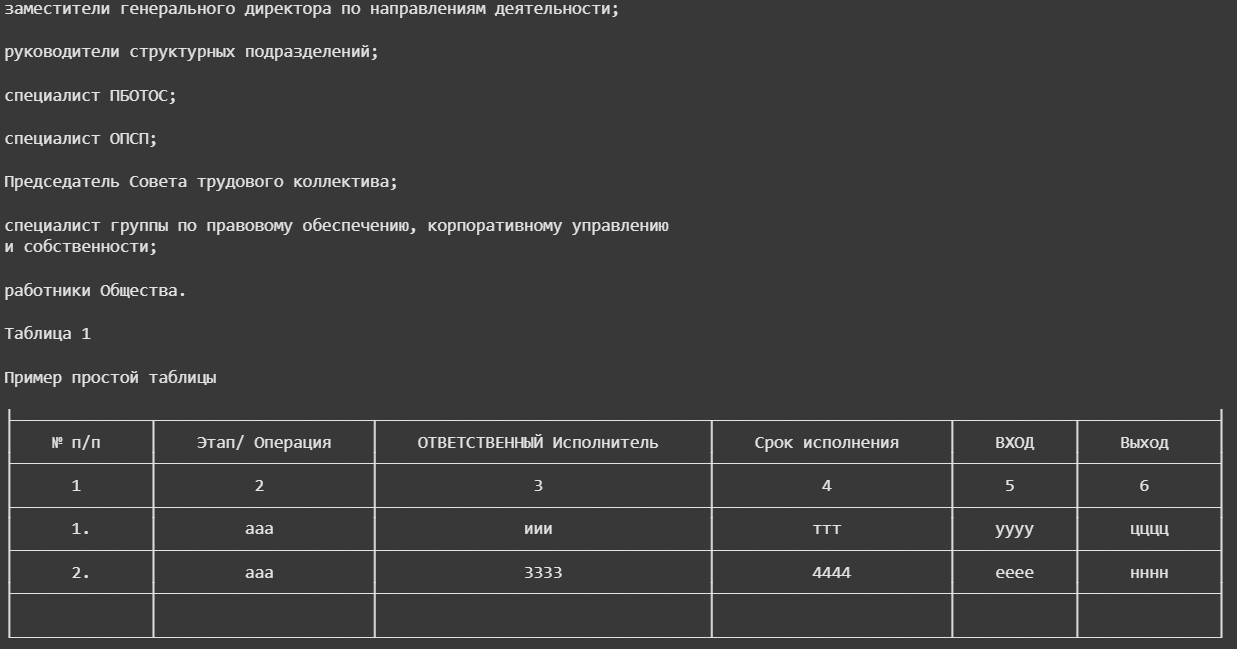


Рисунок 19 — Объединенный текст и таблица