В данной работе мы поработаем с PDF файлами используя Python, а именно: мы рассмотрим, как извлекать текст, изображения, разделять на страницы, найти все страницы файла, где имеется нужный нам текст, как вставлять изображения в этот файл, удалять страницы, разделять страницы на четные и нечетные. Давайте же начнем!

**Извлечение текста с помощью PyPDF2 и PyMuPDF**

Сначала сделаем извлечение текста двумя методами. Первый – используя библиотеку PyPDF2, а второй – PyMuPDF. Что это вообще за библиотеки? PyPDF2 – это библиотека для извлечения информации и содержимого документов, постраничного разделения документов, объединения документов, обрезки страниц и добавления водяных знаков. А PyMuPDF (известный как fitz) - привязка Python для MuPDF, который является облегченным средством просмотра PDF и XPS. Именно поэтому, первым делом мы устанавливаем эти библиотеки: pip3 install pypdf2, pip3 install pymupdf. Далее, в папке с проектом мы создаем еще три дополнительные папки: images, source и dist. Папки images и dist будем использовать для записи результатов работы своих программ, а в папке source храним исходные PDF‑файлы (которые надо будет заранее туда положить), сами скрипты будем хранить в корне. После всех этих действий, приступаем к извлечению текста с помощью PyPDF2:

from PyPDF2 import PdfFileReader

pdf\_document = "source/YourFile.pdf"

with open(pdf\_document, "rb") as filehandle:

pdf = PdfFileReader(filehandle)

info = pdf.getDocumentInfo()

pages = pdf.getNumPages()

print("Количество страниц в документе: %i\n\n" % pages)

print("Мета-описание: ", info)

for i in range(pages):

page = pdf.getPage(i)

print("Стр.", i, " мета: ", page, "\n\nСодержание;\n")

print(page.extractText())

В данном коде мы импортируем PdfFileReader, помня о том, что пакет уже установлен. Задаём имя файла из папки source, открывает документ и получаем информацию о документе, используя метод getDocumentInfo() и общее количество страниц getNumPages(). Далее в цикле for читаем каждую страницу, получаем содержимое page.extractText() и печатаем в stdout. Обратите внимание, что PyPDF2 начинает считать страницы с 0, и поэтому вызов pdf.getPage(i) при i = 0 извлекает первую страницу документа. Результат:

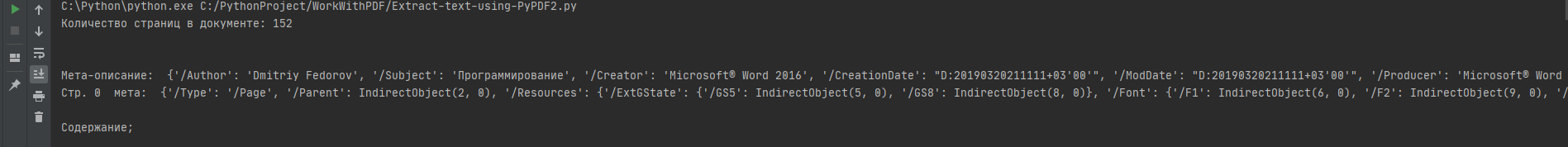


Рисунок 1 – Извлечение текста с помощью PyPDF2

Если использовать библиотеку PyMuPDF, то код выполняется аналогично предыдущему методу, единственный момент заключается в том, что импортируемый модуль имеет имя fitz, что соответствует имени PyMuPDF в ранних версиях:

import fitz

pdf\_document = "./source/ YourFile.pdf "

doc = fitz.open(pdf\_document)

print("Исходный документ: ", doc)

print("\nКоличество страниц: %i\n\n------------------\n\n" % doc.pageCount)

print(doc.metadata)

for current\_page in range(len(doc)):

page = doc.loadPage(current\_page)

page\_text = page.getText("text")

print("Стр. ", current\_page+1, "\n\nСодержание;\n")

print(page\_text)

Приятной особенностью PyMuPDF является то, что он сохраняет исходную структуру документа без изменений — целые абзацы с разрывами строк сохраняются такими же, как в PDF‑документе. Результат:

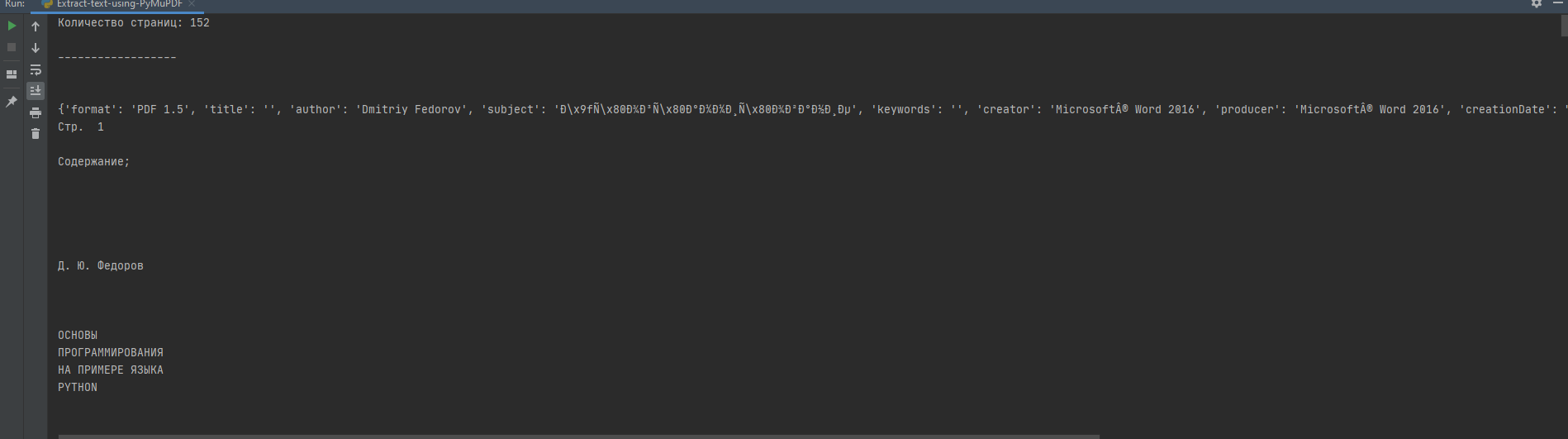


Рисунок 2 – Извлечение текста с помощью PyMuPDF

## Извлечение изображений из PDF с помощью PyMuPDF

## Переходим к изображениям. PyMuPDF упрощает извлечение изображений из документов PDF с использованием метода getPageImageList(). Скрипт, приведённый ниже, основан на примере из вики-страницы PyMuPDF и извлекает и постранично сохраняет все изображения из PDF в формате PNG. Если изображение имеет цветовое пространство CMYK, оно будет сначала преобразовано в RGB. При этом, все извлеченные изображения будут сохраняться у нас в папку images. Сам код:

import fitz

pdf\_document = "source/ YourFile.pdf "

doc = fitz.open(pdf\_document)

print("Исходный документ", doc)

print("\nКоличество страниц: %i\n\n------------------\n\n" % doc.pageCount)

print(doc.metadata)

page\_count = 0

for i in range(len(doc)):

for img in doc.getPageImageList(i):

xref = img[0]

pix = fitz.Pixmap(doc, xref)

pix1 = fitz.Pixmap(fitz.csRGB, pix)

page\_count += 1

pix1.writePNG("images/picture\_number\_%s\_from\_page\_%s.png" % (page\_count, i+1))

print("Image number ", page\_count, " writed...")

pix1 = None

## В моем случае, код извлек из PDF файла 244 изображения. И все это произошло меньше чем за минуту! Результат:

## 

## Рисунок 3 – Извлечение изображений из PDF файла

## Разделение PDF‑файлов на страницы с помощью PyPDF2

Для этого примера, в первую очередь необходимо импортировать классы PdfFileReader и PdfFileWriter. Затем мы открываем файл PDF, создаем объект для чтения и перебираем все страницы, используя метод объекта для чтения getNumPages. Внутри цикла for мы создаем новый экземпляр PdfFileWriter, который еще не содержит страниц. Затем мы добавляем текущую страницу к нашему объекту записи, используя метод pdfWriter.addPage(). Этот метод принимает объект страницы, который мы получаем, используя метод PdfFileReader.getPage(). Следующим шагом является создание уникального имени файла, что мы делаем, используя исходное имя файла плюс слово «page» плюс номер страницы. Мы добавляем 1 к текущему номеру страницы, потому что PyPDF2 считает номера страниц, начиная с нуля. Наконец, мы открываем новое имя файла в режиме (режиме wb) записи двоичного файла и используем метод write() класса pdfWriter для сохранения извлеченной страницы на диск. Все извлеченные страницы запишутся в папку dist. Сам код:

from PyPDF2 import PdfFileReader, PdfFileWriter

pdf\_document = "source/ YourFile.pdf "

pdf = PdfFileReader(pdf\_document)

for page in range(pdf.getNumPages()):

pdf\_writer = PdfFileWriter()

current\_page = pdf.getPage(page)

pdf\_writer.addPage(current\_page)

outputFilename = "dist/Computer-Vision-Resources-page-{}.pdf".format(page + 1)

with open(outputFilename, "wb") as out:

pdf\_writer.write(out)

print("created", outputFilename)

Результат:

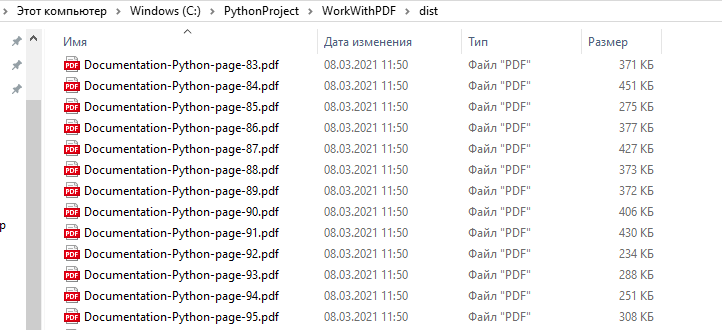


Рисунок 4 – Разделение PDF файла на страницы

## Найти все страницы, где есть заданный текст

## Этот скрипт довольно практичен и работает аналогично pdfgrep. Используя PyMuPDF, скрипт возвращает все номера страниц, которые содержат заданную строку поиска. Страницы загружаются одна за другой и с помощью метода searchFor() обнаруживаются все вхождения строки поиска. В случае совпадения соответствующее сообщение печатается на stdout. В моем случае я нашел все страницы, которые содержат слово “Python”:

import fitz

filename = "source/ YourFile.pdf "

search\_term = "COMPUTER VISION"

pdf\_document = fitz.open(filename)

for current\_page in range(len(pdf\_document)):

page = pdf\_document.loadPage(current\_page)

if page.searchFor(search\_term):

print("%s найдено на странице %i" % (search\_term, current\_page+1))

## 

## Рисунок 5 – Результаты поиска Python

## Добавление водяного знака с помощью PyPDF2

В пакете PyPDF2 есть метод mergepage(), который использует другой PDF в качестве водяного знака или штампа. Сам PDF файл, который мы хотим, чтобы программа использовала в качестве водяного знака кладем в папку source (в моем случае это mshe-logo-512x512.pdf). Начнем с чтения первой страницы исходного PDF‑документа и водяного знака. Для чтения файла мы используем класс PdfFileReader(). На втором шаге эти две страницы объединяются с помощью метода mergepage(), после чего результат записывается в выходной файл. Эти три этапа — создание объекта на основе класса PdfFileWriter(), добавление объединенной страницы к этому объекту с помощью метода addPage() и запись нового контента на выходную страницу с помощью метода write(). При этом, получившийся новый PDF файл с водяным знаком появится в папке dist. Сам код:

# Добавление водяного знака в одностраничный PDF

import PyPDF2

input\_file = "source/ YourFile.pdf "

output\_file = "dist/ YourFile -page-drafted.pdf"

watermark\_file = "source/mshe-logo-512x512.pdf"

with open(input\_file, "rb") as filehandle\_input:

# читать содержимое исходного файла

pdf = PyPDF2.PdfFileReader(filehandle\_input)

with open(watermark\_file, "rb") as filehandle\_watermark:

# читать содержание водяного знака

watermark = PyPDF2.PdfFileReader(filehandle\_watermark)

# получить первую страницу оригинального PDF

first\_page = pdf.getPage(0)

# получить первую страницу водяного знака PDF

first\_page\_watermark = watermark.getPage(0)

# объединить две страницы

first\_page.mergePage(first\_page\_watermark)

# создать объект записи PDF для выходного файла

pdf\_writer = PyPDF2.PdfFileWriter()

# добавить страницу

pdf\_writer.addPage(first\_page)

with open(output\_file, "wb") as filehandle\_output:

# записать файл с водяными знаками в новый файл

pdf\_writer.write(filehandle\_output)

Результат:

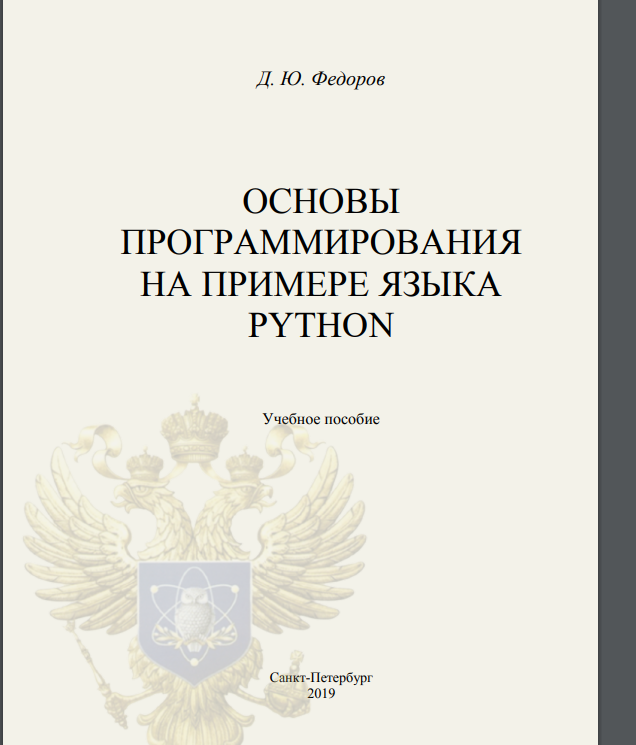


Рисунок 6 – Результат добавления водяного знака в одностраничный PDF

## Добавление изображения с помощью PyMuPDF

## В данном коде, я добавил в мой PDF файл изображение, которое предварительно положил в папку source (в моем случае это image.jpg). Сам код:

import fitz

input\_file = "source/ YourFile.pdf "

output\_file = "dist/ YourFile -page-image.pdf"

barcode\_file = "source/YourImage.jpg"

# определить позицию (верхний правый угол)

image\_rectangle = fitz.Rect(450, 170, 550, 270)

# retrieve the first page of the PDF

file\_handle = fitz.open(input\_file)

first\_page = file\_handle[0]

# добавить изображение

first\_page.insertImage(image\_rectangle, filename=barcode\_file)

file\_handle.save(output\_file)

Положение изображения определяется как rectangle (прямоугольник) методом fitz.Rect(), который требует двух пар координат — левый верхний угол (x1,y1) и правый нижний угол (x2,y2) изображения. В PyMuPDF левому верхнему углу страницы соответствуют координаты (0,0). После открытия входного файла и извлечения из него первой страницы с помощью метода insertImage() добавляется наше изображение. Этот метод требует двух параметров — позиционирование с использованием imageRectangle и имя файла изображения для вставки. С помощью метода save() измененный PDF‑файл сохраняется на диске. Результат:

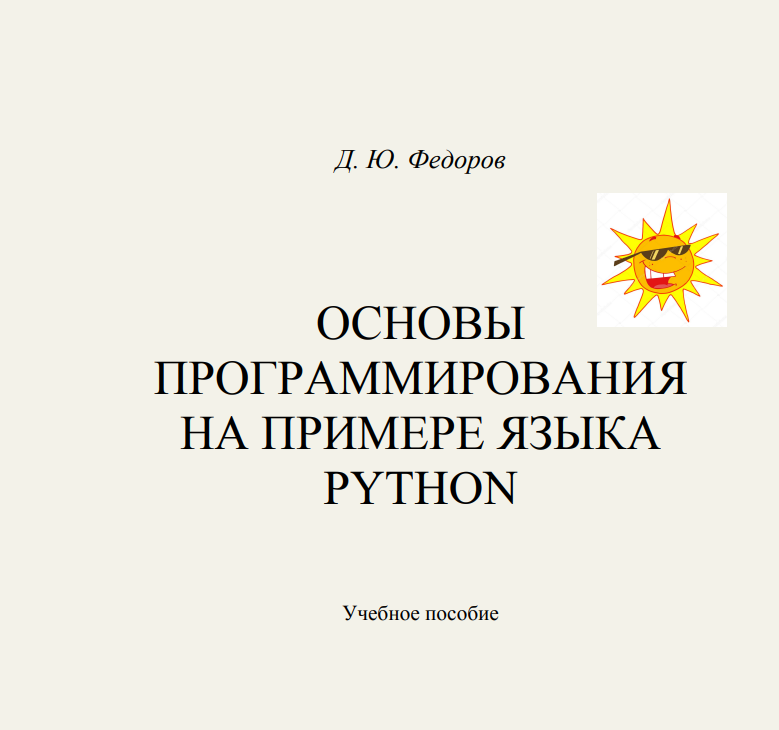


Рисунок 7 – Добавление изображения на начальную страницу в исходный PDF

## Добавление штампов с pdfrw

## [pdfrw](https://github.com/pmaupin/pdfrw.git) — это библиотека Python и утилита, которая читает и записывает PDF‑файлы. И перед тем как выполнять данное задание, эту библиотеку необходимо будет установить. После этого, мы из данного пакета импортируем три класса — PdfReader PdfWriter и PageMerge. Устанавливаем соответственно объекты чтения/записи, для доступа как к содержимому PDF, так и к нашему изображению. Для каждой страницы в исходном документе вы продолжаем создавать объекты PageMerge, к которому добавляем водяной знак и который отображается всё это с помощью метода render(). Сам код:

from pdfrw import PdfReader, PdfWriter, PageMerge

input\_file = "source/ YourFile.pdf "

output\_file = "dist/ YourFile-pages-image.pdf"

watermark\_file = "source/mshe-logo-512x512.pdf "

# определяем объекты чтения и записи

reader\_input = PdfReader(input\_file)

writer\_output = PdfWriter()

watermark\_input = PdfReader(watermark\_file)

watermark = watermark\_input.pages[0]

# просматривать страницы одну за другой

for current\_page in range(len(reader\_input.pages)):

merger = PageMerge(reader\_input.pages[current\_page])

merger.add(watermark).render()

# записать измененный контент на диск

writer\_output.write(output\_file, reader\_input)

## Результат:

## 

## Рисунок 8 – Добавление изображения на все страницы исходного документа

## Удаление страниц с помощью PyMuPDF

Библиотека PyMuPDF включает в себя довольно много сложных методов, которые упрощают удаление страниц из файла PDF. Он позволяет вам указать либо одну страницу (метод deletePage()), либо диапазон номеров страниц (метод deletePageRange()), либо список с номерами страниц (метод select()). В следующем примере будет показано, как использовать список для выбора страниц, которые следует сохранить из исходного документа. Имейте в виду, что страницы, которые не указаны, не будут частью выходного документа. В нашем случае выходной документ содержит только первую, вторую и четвертую страницы:

import fitz

input\_file = "source/YourFile.pdf "

output\_file = "dist/YourFile -rearranged.pdf"

# Определите страницы для сохранения - 1, 2 и 4

file\_handle = fitz.open(input\_file)

pages\_list = [0,1,3]

# Выберите страницы и сохраните вывод

file\_handle.select(pages\_list)

file\_handle.save(output\_file)

Результат:

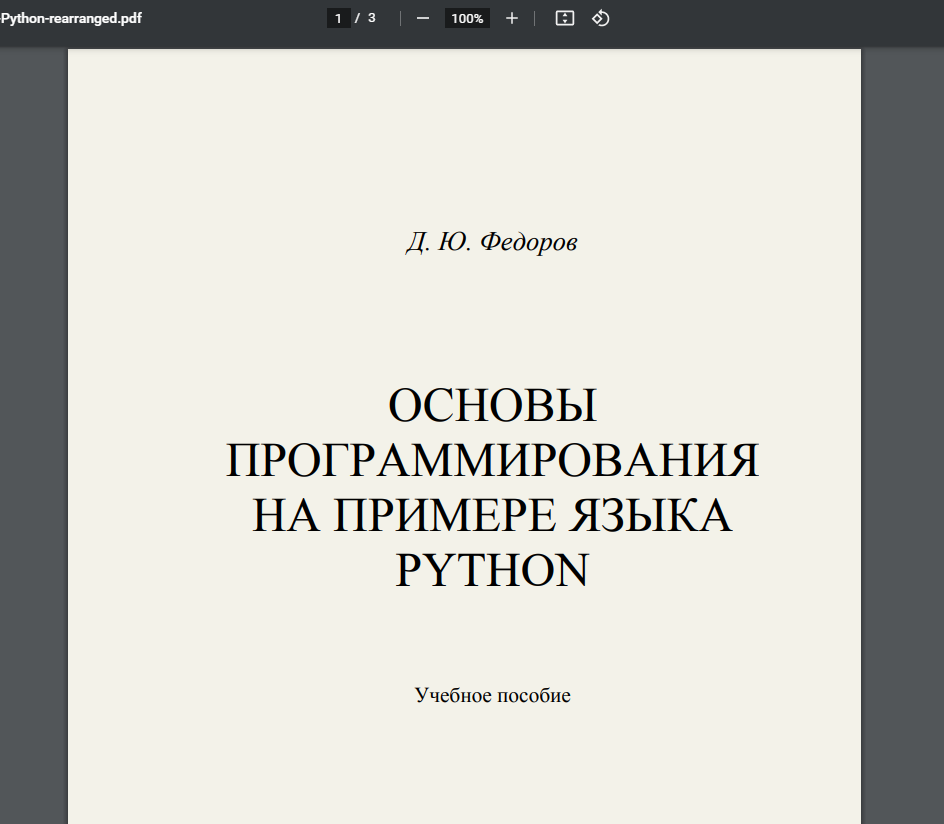


Рисунок 9 – Удаление всех страниц файла, кроме 1,2 и 4

## Разделение четных и нечетных страниц с помощью PyPDF2

Следующий пример использует PyPDF2 и разделяет файл на четные и нечетные страницы, сохраняя четные страницы в файле Documentation-Python-even.pdf и нечетные страницы в Documentation-Python-odd.pdf. Этот скрипт Python начинается с определения двух выходных файлов, Documentation-Python-even.pdf и Documentation-Python-odd.pdf, а также соответствующие им объекты для записи pdf\_writer\_even и pdf\_writer\_odd. Затем в цикле for скрипт просматривает весь файл PDF и читает одну страницу за другой. Страницы с четными номерами страниц добавляются в поток pdf\_writer\_even с помощью addPage(), а нечетные номера добавляются в поток pdf\_writer\_odd. В конце два потока сохраняются на диск в отдельных файлах, как определено ранее. Сам код:

from PyPDF2 import PdfFileReader, PdfFileWriter

pdf\_document = "source/ YourFile.pdf "

pdf = PdfFileReader(pdf\_document)

# Выходные файлы для новых PDF-файлов

output\_filename\_even = "dist/ YourFile -even.pdf"

output\_filename\_odd = "dist/ YourFile -odd.pdf"

pdf\_writer\_even = PdfFileWriter()

pdf\_writer\_odd = PdfFileWriter()

# Получить досягаемую страницу и добавить ее в соответствующую

# выходной файл на основе номера страницы

for page in range(pdf.getNumPages()):

current\_page = pdf.getPage(page)

if page % 2 == 0:

pdf\_writer\_odd.addPage(current\_page)

else:

pdf\_writer\_even.addPage(current\_page)

# Записать данные на диск

with open(output\_filename\_even, "wb") as out:

pdf\_writer\_even.write(out)

print("created", output\_filename\_even)

# Записать данные на диск

with open(output\_filename\_odd, "wb") as out:

pdf\_writer\_odd.write(out)

print("created", output\_filename\_odd)

## 