【Python×PDF】PyPDF2によるPDFファイルの結合・分割と画像抽出する方法

2022.10.11 2021.03.31

【注意】

本記事で取り上げる、「PyPDF2ライブラリ」は、2022年に大幅に刷新されました。本記事は、旧バージョンのPyPDF2の使い方を解説しています。
<https://pypdf2.readthedocs.io/en/latest/meta/history.html#history-of-pypdf2>

大きな使い方の概要やフローはそのまま流用できますが、クラス名やオブジェクト名に変更が入っており、記事中のサンプルコードなどはそのままでは実行できませんので注意してください。

現在、新バージョンに対応する解説を執筆しておりますので、お待ちください。

私たちが普段使っている利便性の高いファイル形式の一つに「**PDF（Portable Document Format）**」があります。

> **文章などのコンテンツをPDF化することで、**
>
> - ✔ **元ファイルの容量を圧縮する**
> - ✔ **元ファイルの内容を書き換えすることができないようする(保護)**
> - ✔ **Readerがインストールされたpcやスマホから簡単に閲覧・検索できる**
>
> などのようなことができるようになります。

今回はこのPDFファイルをPythonで操作する方法を紹介していきます。

PDFの「利便性と汎用性」と、Pythonの「拡張性、データ分析・処理など」それぞれの得意とする機能を掛け合わせることでさら活用の幅が広がることでしょう。

この記事で学べること

- ☑ **多量のPDFファイルを一括処理して作業の手間を減らす。**
- ☑ **PDF内のテキストや画像を抽出して分析する。**
- ☑ **抽出したコンテンツから新たなPDFファイルの作成・編集をする。**

図1. PDFとPythonを組み合わせてより便利に

PythonからPDFファイルを操作するには専用の外部ライブラリをインストール・インポートする必要があります。
PDFを操作するライブラリには、`PDFMiner`, `PyPDF2`, `ReportLab` といったものなど、いくつか存在します。

ただし、**PDFは非常に複雑な仕様となっているので一つのライブラリで全ての機能をカバーすることは現状ではできないようです。**そのため、それぞれのライブラリには特徴や得意とする操作が異なります。**ユーザーは目的に応じて使い分ける、もしくは組み合わせて使う必要があります。**

そこで、目的や特徴によってどのように使い分けるかといった大まかな目安を次のようにまとめてみましたので参考にして下さい。

ライブラリ	用途・特徴	その他
PDFMiner	テキストを抽出する	日本語に対応, pipにて導入
PyPDF2	画像を抽出する・PDFファイルの結合や分割処理	日本語に非対応,pip/GitHubにて導入
ReportLab	PDFを新規作成する(テキスト・図表・グラフ)	低水準のライブラリ※, pipにて導入

※性能が低いというわけではなく、細かく指定できる(指定しなければならない)

表1. PDFを操作する外部ライブラリの比較

PDFMinerによるPDFの操作する方法については次の記事で取り上げていますので参照して下さい。

参考記事

【Python×PDF】PDFMinerライブラリでPDFからテキストを抽出方法【徹底解説】

この記事では「PDFMiner」ライブラリで、PDFファイルからテキスト(文章)コンテンツを抽出する方法を解説しています。ライブラリの紹介からインストール方法、実践まで参考になります。

www.shibutan-bloomers.com 2021.03.23

さて今回の記事ではこれらのライブラリの中から「**PyPDF2**」を使ってPDFファイルを操作する方法について紹介していきます。

具体的には、①「PDFに埋め込まれた画像（JPEG,PING形式）を抽出する方法」や、②「複数のPDFファイル同士を結合する方法」、またその逆に③「ページごとにファイルを分割する手順」について図解を交えてながら分かりやすく解説します。

また、開発環境としては、パッケージ管理ツール<Anaconda>が導入済みであることを前提としております。

記事内で動作確認した開発環境とバージョン情報は次のとおりとなります。異なる環境・バージョンのライブラリを使う際はこの点、ご留意ください。

- ✔ **Python 3.10.6(64bit)**
- ✔ **PyPDF2 1.26.0**
- ✔ **JupyterNotebook 6.4.8**

Contents

1. PDFを操作するライブラリ（PyPDF2）の概要

PyPDF2は「pyPdf」というライブラリの機能拡充版として開発されたPDFファイルを操作するモジュールとしてよく使われています。
日本語のテキストに非対応という我々日本人にとっては残念な部分もありますが、コンテンツ(テキスト・画像)を抽出する、PDFの結合や分割処理といったことを簡単なコードで実現できます。

1.1 インストールと動作確認

2022年のライブラリのリニューアルに伴い、インストール方法に変更が入っています。
2022年現在、以下インストーラはダウンロード不可となっています。
新Verでは、Pythonコマンドの「pip」対応が可能となっており、手動インストールの必要がなくなりました。

PyPDF2はAnacodaには同梱されていないので別途インストールする必要があります。
Pythonのライブラリ管理ツール「pip」でインストールする方法もありますが、画像の抽出機能に一部欠点があるようです。そこで、**今回は有志により開発がすすめられGitHub上で公開されているインストーラをダウンロードして導入する方法を採ること**にします。

GitHubのアクセス先は以下のとおりで、 Clone or download → Download ZIP と進み「**PyPDF2-master.zip**」を入手します。

> **GitHub <PyPDF2> ダウンロード先**
> https://github.com/mstamy2/PyPDF2

入出したアーカイブを解凍したディレクトリに移動して、「setup.py」を次のコマンドをAnaconda Promptなどで実行します。

```
python setup.py install
```

PythonスクリプトでPyPDF2モジュールをインポートしてエラーが表示されなければインストールは成功です。

```
import PyPDF2
```

1.2 PyPDF2のクラスの概要

PyPDF2が提供するクラスは主として「**PdfFileReader**」「**PageObject**」「**PdfFileMerger**」「**PdfFileWrite**」の4つがあります。

それぞれ、PDFオブジェクトの「**ファイルの読込と全般**」「**コンテンツの抽出**」「**ファイル同士の結合**」「**ページの追加と削除**」といったように機能別にクラス分けされており、PDFMinerよりもシンプルで直感的に理解しやすいクラス構造になっている印象です。（図２）

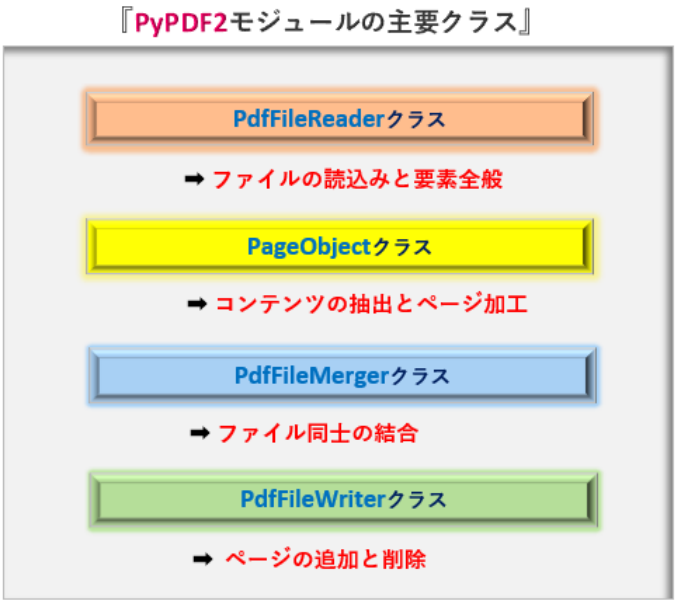


図２. PyPDF2の主要な３つのクラス

各クラスには多くのメソッド・プロパティが用意されていますので単独で使うこともできますし、目的によっては複数のクラスを組み合わせていきます。

PyPDF2はコンテンツをゼロから全て作るといったことは得意ではありません。よって、まずは既存のPDFから個別ページ情報(PageObjectオブジェクト)を読込んで、編集処理することを前提とすることが多いです。(図4)

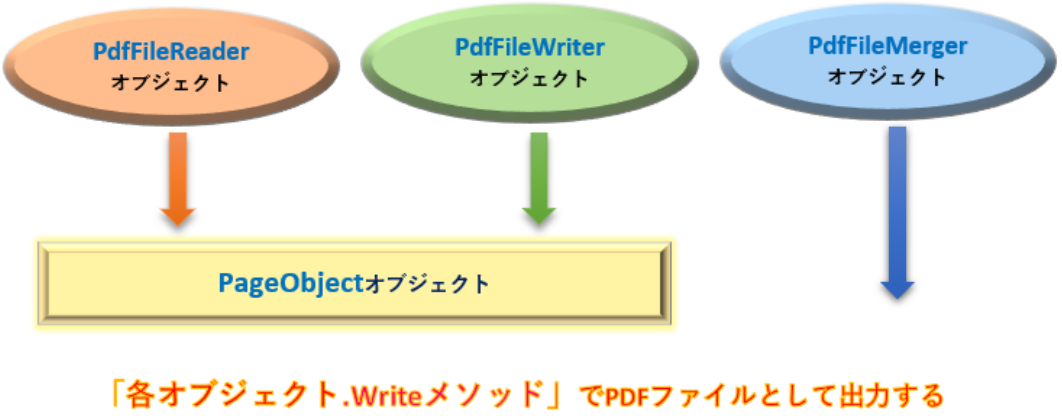


図3. オブジェクトの階層構造

次章からは各クラスの主な仕様とサンプルコードによる活用例を紹介していきます。

本サイトでの各種クラスなどの使い方は一例です。省略可能なオプション引数などについては割愛していますので、詳細や不明点などは必要に応じて公式サイトを適宜参照してください。

公式サイト: **https://pythonhosted.org/PyPDF2/index.html**

2. PdfFileReaderクラス

PDFファイルを読込む、また構成情報を取得するためのクラスです。**PyPDF2は既存のPDFファイルを解析・編集することを得意とするモジュールですので、まずはこの「PdfFileReaderオブジェクト」を起点とすることが多い**です。

2.1 PdfFileReaderオブジェクトと関連メソッド

PDFから情報を取り込むには `PdfFileReaderクラス` からオブジェクトを取得した後、その配下の関数(メソッド・プロパティ)を使うことになります。

PdfFileReaderオブジェクトは、次の書式のように `引数:stream` に対象PDFのファイルオブジェクト(“**b**“バイナリモード)を指定して取得します。

PdfFileReaderオブジェクト

from PyPDF2 import PdfFileReader
PdfFileReader(stream, strict, overwriteWarnings)

引数: stream :PDFのファイルオブジェクト ファイルオブジェクトは**"rb"**:バイナリの読取専用で取得する
引数: strict：致命的なエラーが発生した場合の告知の有無(デフォルト:**True**)　**True**:(告知する)/**False**:(告知しない)
引数: overwriteWarnings：上書き保存の警告発呼の有無(デフォルト:**True**)　**True**:(告知する)/**False**:(告知しない)

戻り値: `PdfFileReaderオブジェクト`

次にPdfFileReaderオブジェクト配下の主なメソッドは、以下のようなものがあります。(表2)

【PDFFileReaderオブジェクト】	【機能】	【その他、詳細】
getPage(pageNumber**)**	Page情報を取得する	`PageObjectオブジェクト`
getNumPages()	ページ数の取得する	整数
decrypt(password**)**	パスワードを解く	文字列
getPageLayout()	ページレイアウトを取得する	"/NoLayout","/Singlepage"など
getOutlines(node, outlines**)**	目次情報を取得する	`Destinationオブジェクト`

表2 PdfFileReaderオブジェクト配下のメソッド

この他にも、「**ページモード**」「**フィールド**」「**メタデータ**」を取得するメソッドや読取専用のプロパティが用意されています。

PDFの個別ページ情報を表す、PageObjectオブジェクトを取得する `getPage()メソッド` とページ総数を取得する `getNumPages()メソッド` は必ず使うことになるので次項にまとめます。

2.1.1 PageObjectオブジェクトを取得する

PDFの個別ページ情報を管理する `PageObjectオブジェクト` は次の書式で取得します。引数には取得したいページ番号を整数で指定します。

PdfFileReaderオブジェクト

PdfFileReaderオブジェクト.getPage(pageNumber)

引数: pageNumber:取得したいページ番号を指定 1ページ目は「0」
戻り値: `PageObjectオブジェクト`

PageObjectオブジェクト中身を確認した結果は図4のようになります。

辞書の多重ネスト構造になっており、ページの内容によってKey<キー>の構成も異なります。また、**Value<値>もリストやさらに別のオブジェクトで構成**されており、**PDFファイルは複雑な構造で定義されていること**が分かります。

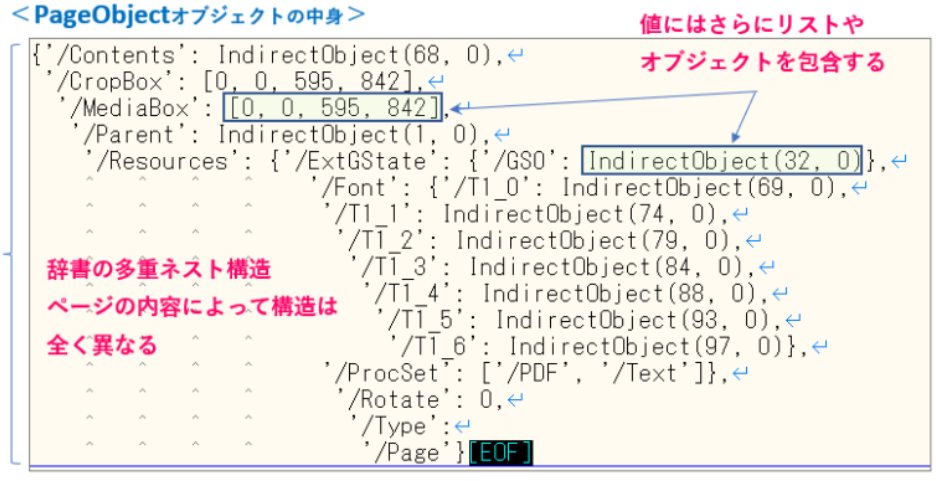


図4. PageObjectオブジェクトの中身

PageObjectオブジェクトにもページ設定にまつわる多くのメソッドやプロパティが用意されています。主なものとしてコンテンツ(文字)の取得、ページの回転・拡大縮小・ページの結合(ファイルごとではなく)などができます。(表3)

【PageObjectオブジェクト】	【機能】	【その他、詳細】
extractText()	文字を抽出する	UniCodeのみ
getContents()	コンテンツを取得する	PDF仕様7.7.3.3参照
mergePage(page2**)**	ページを結合する	結合するPageObjectを指定
rotateClockwise(angle**)**	ページの回転させる	90度づつ時計回り
scale(sx, sy**)**	ページを拡大・縮小する	1.0以下の実数を指定

表3. PageObjectオブジェクト配下のメソッド

2.1.2 PDFファイルのページ数を取得する

PDFファイルのページ数を取得するには、`getNumPages()メソッド` もしくは、`numPagesプロパティ` (取得専用)を使います。
基本的にPyPDF2では1ページずつ個別に処理をしていくため、あらかじめPDFの構成ページ数を把握しておく必要がありますのでこのメソッドを使うことになるでしょう。

PdfFileReaderオブジェクト

PdfFileReaderオブジェクト.getNumPages()

引数: なし
戻り値: **ページ数**

またパスワードなどで制限がかかっている場合は、エラー(”**PdfReadError**”)が発生します。

2.2 JPEG/PNG画像の抽出と保存

PdfFileReaderクラスのオブジェクトやメソッドの活用事例を簡単なサンプルコードで確認してみましょう。コードの概要は以下の通りです。

①. PdfFileReaderオブジェクトから、個別ページ情報にアクセスし、辞書形式の多重ネスト構造を解析する。
②. JPEG/PING画像データが格納されたKey<キー>を探索、(図5)そこから画像本体のバイナリデータを取得して、名前を付けて保存する。

JPEG/PING画像のバイナリデータが置かれている階層は次のようになります。(図5)

例えば、JPEG画像が格納された階層を探索する場合には、`”/Resorces”`<key> →`”/Xobject”`<key>配下の要素のオブジェクトを取得、そこからさらに、オブジェクト内を探索て、“**Image**”や”**FlateDecode**”(JPEG画像)という値を見つけることができたら、そこのバイナリデータがJPEG画像データということです。

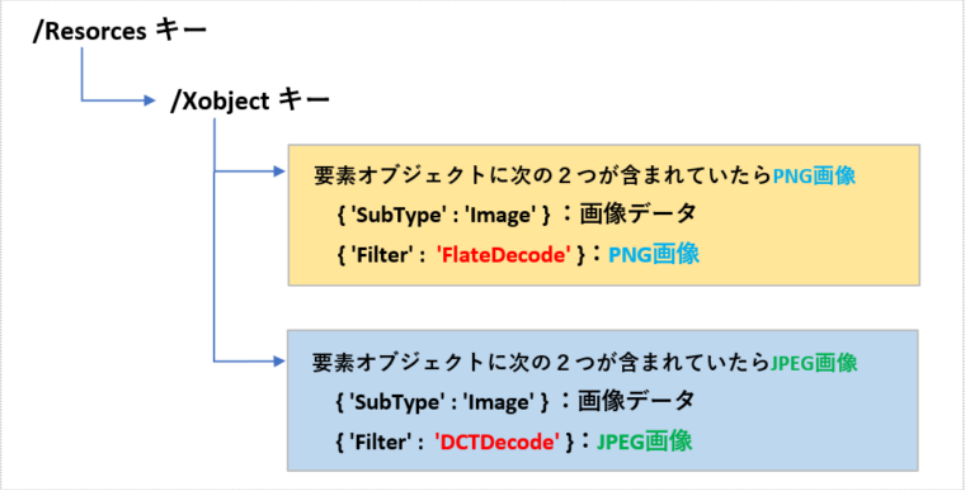


図5. JPEG/PNG画像ファイルのデータ階層構造

まずは、JPEG画像を抽出するコードから紹介します。（List1）

List1

```
from PyPDF2 import PdfFileReader  # PdfFileReaderクラスのインポート

fp = open("PDF_test.pdf", 'rb')   # PDFファイルのFileオブジェクトを取得する(読込み・バイナリモード)
reader = PdfFileReader(fp)        # Fileオブジェクトを引数にしてPdfFileReaderオブジェクトを取得

pgnum = reader.getNumPages()      # PdfFileReaderオブジェクトのgetNumPagesメソッドでページ数を取得

for i in range(pgnum):
    pg = reader.getPage(i)        # PdfFileReaderオブジェクトのgetPageメソッドでPDFPageオブジェクトを取得

 # 以降、Pageオブジェクトの解析・探索----------------------------------------------------↓ここから

    if '/XObject' in pg['/Resources']:  # (1)
        xobjs = pg['/Resources']['/XObject']

        for key, obj in xobjs.items():
            item = obj.getObject()

            if item['/Subtype'] == '/Image':   # (2)
                if '/Filter' in item and item['/Filter'] == '/DCTDecode':   # (3)
 # ------------------------------------------------------------------------------------↑ここまで

                    data = item.getData()           #  PDFPageオブジェクトからJPEGファイルのバイナリデータを取得
                    img = open(key[1:]+".jpg", 'wb') #  出力ファイル用のFileオブジェクトを取得(書込み・バイナリモード)

                    img.write(data)                 #  Fileオブジェクトへ JPEGファイルのデータを書き込む
                    img.close()
fp.close()
```

それでは、ポイントを解説します。
(解説中の(1)(2)(3)はコード内のコメントに対応しています。)

1,3,4行目：【PdfFileReaderオブジェクトの取得】

1行目で `PdfFileReaderクラス` をインポートしています。3行目で `open()関数` のモードを'rb'(バイナリ読み取り)に指定してFileオブジェクトを、4行目でそれを引数にしてPdfFileReaderオブジェクトを取得しています。

6,9行目：【PageObjectオブジェクトの取得】

6行目の `getNumPages()メソッド` で取得したページ数分のPageObjectオブジェクトを9行目の `getPage()メソッド` で取得しています。

12~22行目：【JPEG画像の探索】

12~22行目でPageObjectオブジェクトの階層構造(辞書形式)を解析しています。
具体的には、(1)でキー/Resources,/XObjectの値(オブジェクト)を取得、
(2)「**/Subtypeキー**」が「**/Image**」でかつ(3)「**/Filterキー**」が「**/DCTDecode**」である要素を探索します。

以降、JPEG画像のバイナリデータをの **getData()関数** で取得、Fileオブジェクトの **write関数** でJPEG画像をとして保存しています。

・　・　・

次にPING画像を抽出するコードを紹介します。<List2>

List2

```
from PyPDF2 import PdfFileReader # PdfFileReaderクラスのインポート
from PIL import Image            # Imageクラスのインポート

fp = open("PDF_test.pdf", 'rb')
reader = PdfFileReader(fp)

pgnum = reader.getNumPages()

for i in range(pgnum):
    pg = reader.getPage(i)

 # 以降、PDFPageオブジェクトの解析・探索-----------------------------------------------↓ここから

    if '/XObject' in pg['/Resources']:  # (1)
        xobjs = pg['/Resources']['/XObject']

        for key, obj in xobjs.items():
            item = obj.getObject()

            if item['/Subtype'] == '/Image':  # (2)
                if '/Filter' in item and item['/Filter'] == '/FlateDecode':  # (3)
 # ------------------------------------------------------------------------------↑ここまで

                    size = (item['/Width'], item['/Height'])
                    data = item.getData()                    # PDFPageオブジェクトからPNGファイルのバイナリデータを取得
                    img = Image.frombytes("RGB", size, data) # PNGファイルはImageクラスのfrombytesメソッドでImageオブジェクトを取

                    img.save(key[1:]+".png") # Imageオブジェクトのsaveメソッドで名前を付けて保存する

fp.close()
```

<List1>とほぼ同じ処理を行っているので差分のみを解説します。

12~22行目：【PNG画像の探索】

JPEG画像と同様にPageObjectオブジェクトの階層構造(辞書形式)を解析しています。PNG画像を抽出するには(3)の「**/Filterキー**」が「**/FlateDecode**」となっている箇所を探索します。

また、PNG画像を扱う場合はPILモジュールから **Image.frombytes()メソッド** を使ってImageオブジェクトを取得し、PING画像として保存します。

参考までに、Imageオブジェクトを使ってPNG,JPEG形式で保存する場合は、次のように書式が異なります。

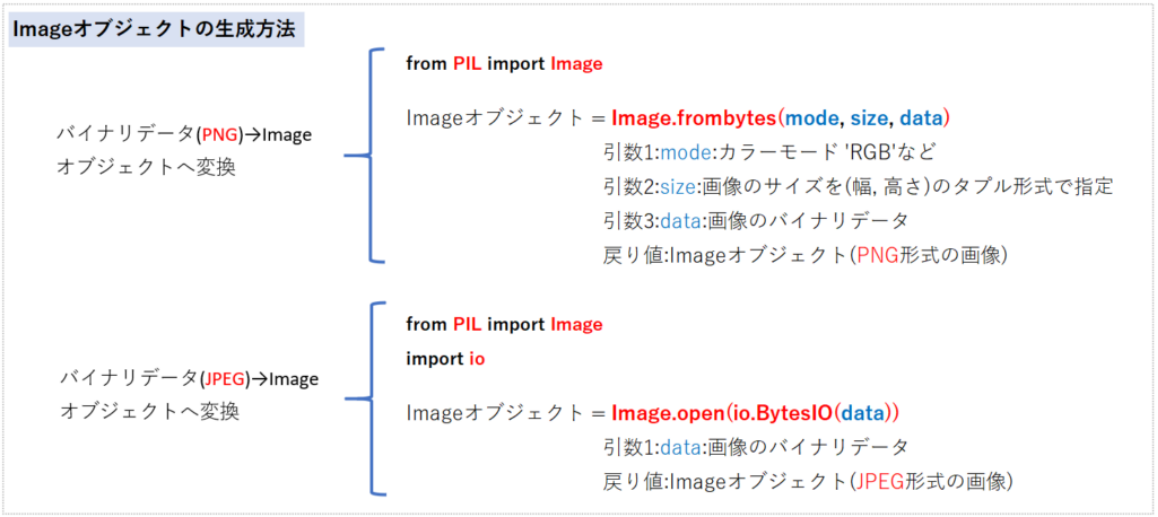


図6. ImageオブジェクトのPNG/JPEG画像の保存

List1/2の実行結果は以下のようになりました。
List1の実行後にはJPEGファイルが、List2の実行後にはPNGファイルが抽出されました。

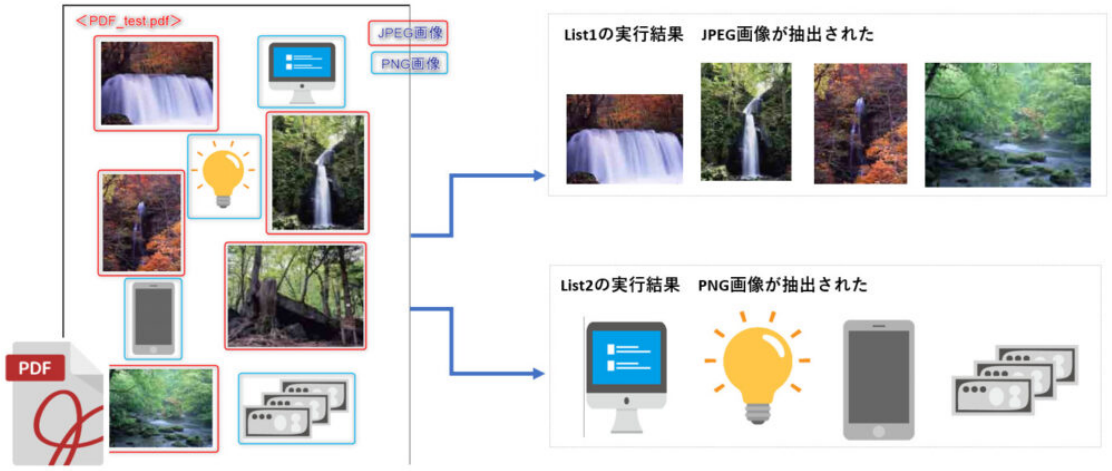


図7. List1/2の実行結果(JPEG,PING画像の抽出)

3. PdfFileMergerクラス

PdfFileMergerクラスは複数のPDFファイルを一つに結合(マージ)する機能を提供するクラスです。**後述するPdfFileWriterクラスを使ってもページ毎の結合はできますが、PdfFileMergerクラスを使えばファイルごとまとめてマージできます。**

3.1 PdfFileMergerオブジェクトと関連メソッド

PDFを結合するには PdfFileMergerクラス からオブジェクトを取得し、配下の関数(メソッド)を使うことになります。PdfFileMergerオブジェクトは、次の書式のように引数指定なしで取得できます。

PdfFileMergerオブジェクト

```
from PyPDF2 import PdfFileMerger
PdfFileMerger()
```

引数: なし
戻り値: PdfFileMergerオブジェクト

PdfFileMergerオブジェクトの取得できたところで、次にオブジェクト以下のマージに関係するメソッドについて解説します。

結合ですから、「**結合元となるPDFファイル**」と「**結合先となるPDFファイル**」があるわけですがその間のやり取りはPdfFileMergerオブジェクトを介して行われることになります。

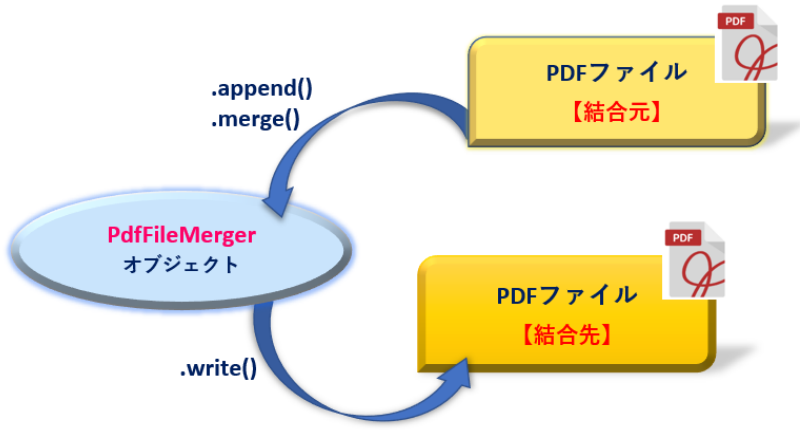


図8. PdfFileMergerオブジェクトの扱い

・　・　・

まず、追加元となるPDFファイルの情報をPdfFileMergerオブジェクトに取り出すメソッドについてです。メソッドには追加先の追加挿入位置により2つに分かれます。一つが、「**結合先ファイルの最後に追加する場合**」そしてもう一つが「**結合先ファイルの任意の位置に挿入する場合**」の２つです。

「結合先ファイルの最後に追加する場合」

結合先のファイルの最後のページ以降に他のファイルを追加結合する場合は append()メソッド を使います。結合するファイルのファイルオブジェクトと対象ページを引数で指定します。イメージは図9のようになります。

append()メソッド

```
PdfFileMergerオブジェクト.append(fileobj, pages)
```

引数: fileobj:結合するPDFのファイルオブジェクト
('rb'モードで読み込まれたファイルオブジェクト)

引数: pages:結合するPDFの対象ページ番号
(PageRange<リンク>, もしくは、**タプル形式(start, stop[, step])**で指定する)

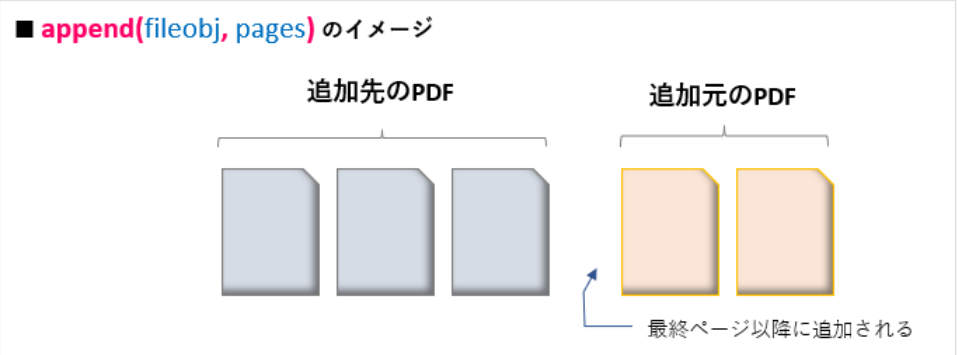


図9. appendメソッドは最終ページ以降に追加して結合する

「結合先ファイルの任意の位置に挿入する場合」

一方、任意のページ位置を指定してファイルを挿入結合する場合には次の `merge()メソッド` を使います。先のappend()メソッドから、挿入位置の指定をする引数positionが追加されています。イメージは図9のようになります。

merge()メソッド

PdfFileMergerオブジェクト.merge(position,fileobj, pages)

引数: position:挿入する位置を指定する　ページを整数で指定
引数: fileobj:結合するPDFのファイルオブジェクト
('rb'モードで読み込まれたファイルオブジェクト)
引数: pages:結合するPDFの対象ページ番号
(PageRange<リンク>, もしくは、**タプル形式(start, stop[, step])**で指定する)

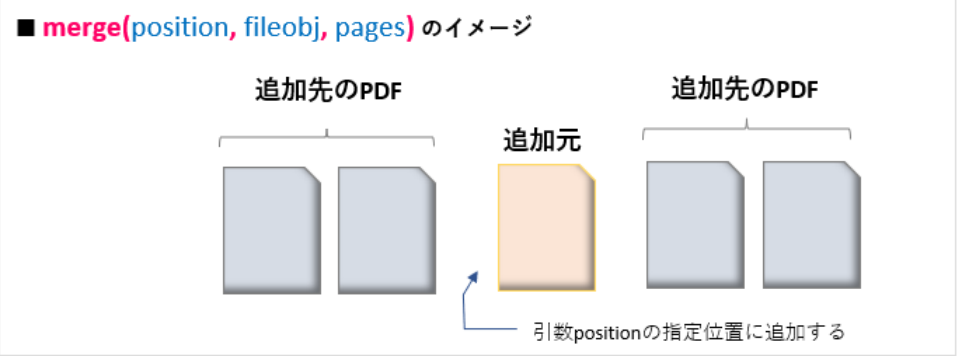


図10. mergeメソッドは任意のページに投入して結合する

次に、追加先となるPDFファイルへPdfFileMergerオブジェクトを書き出す `write()メソッド` です。
引数に指定する出力先のファイルオブジェクトですが、今度は編集・バイナリモード('**rb**')で取得します。また、結合先は「新規ファイル」・「既存ファイル」どちらでも構いません。

write()メソッド

PdfFileMergerオブジェクト.write(fileobj)

引数: fileobj **:**出力先となるPDFのファイルオブジェクト　"**wb**'モード
戻り値: `PDFファイル`

以上が、PdfFileMergerクラスで抑えるべき主要メソッドの紹介です。繰り返しますが、本クラスは複数のファイルを効率的に結合(マージ)することができます。
次項では具体的な活用事例をサンプルコードで紹介します。

3.2 PDFファイルを結合するサンプルコード

ここで、PdfFileMergerクラスの活用例をサンプルコードとともに紹介します。
コードの概要は、２つのPDFファイルを１つに結合して新規のPDFファイルを作成するものです。

List3

```
from PyPDF2 import PdfFileMerger   # PdfFileMergerクラスのインポート

fp_in1 = open('PDF_sample1.pdf', 'rb')   # 1つ目の既存PDFファイルのFileオブジェクトを'rb'モードで取得
fp_in2 = open('PDF_sample2.pdf', 'rb')   # 2つ目の既存PDFファイルのFileオブジェクトを'rb'モードで取得

mgr = PdfFileMerger() # PdfFileMergerオブジェクトの取得
mgr.append(fp_in1)    # Fileオブジェクトを追加
mgr.append(fp_in2)    # Fileオブジェクトを追加

fp_out = open("pdf12.pdf", 'wb')  # 新規Fileオブジェクトを'wb'モードで取得
mgr.write(fp_out)                 # PdfFileMergerオブジェクトをFileオブジェクトに書き出し
fp_out.close()

fp_in1.close()
fp_in2.close()
```

それでは、ポイントを解説します。

1,6行目：【PdfFileMergerのオブジェクトの取得】

PdfFileMergerクラスをインポートし、6行目で **PdfFileMerger**のオブジェクト を取得しています。

3,4,7,8行目：【結合元ファイルの読込み】

7,8行目の **append()**メソッド の引数に結合元となるPDFのFileオブジェクト(“rb”モード)を設定しPdfFileMergerのオブジェクトに追加します。

11行目：【出力先ファイルへ書き出し】

write()メソッド でPdfFileMergerオブジェクトの内容を、新規PDFファイルのFileオブジェクト(“wb”モード)に書き込んで結合されたPDFファイルを出力します。

以降、展開したファイルオブジェクトを閉じるなど後処理をしてコードを終了します。

以上、<List3>を実行した結果を図11に示します。
2つのPDFファイル(PDF_sample1.pdf, PDF_sample2.pdf)を結合して新規PDFファイルが作成されました。

図11. List3の実行結果

4. PdfFileWriterクラス

PdfFileWriterクラス はファイルの構成を編集する機能を提供します。

PyPDF2はコンテンツ(テキスト・画像の挿入)をゼロから作ることは得意ではありません(できません)。既存のPDFファイルのページ構成を入れ替える、コピーするなど構成を編集することでPDFファイルを新規作成します。

4.1 PdfFileWriterオブジェクトと関連メソッド

PDFの編集には PdfFileWriterクラス のオブジェクトと、その配下のメソッドを使います。
PdfFileWriterオブジェクトは、次の書式のように引数指定なしでインスタンスを生成します。

PdfFileWriterオブジェクト

from PyPDF2 import PdfFileWriter
PdfFileWriter()

引数: なし
戻り値: PdfFileWriterオブジェクト

PdfFileWriterオブジェクトの取得ができたところで、次にオブジェクト以下の主要なメソッドについて解説します。

既存ファイルを編集しますので、「**参照元となるPDFファイル**」と「**編集先となるPDFファイル**」があるわけですがその間のやり取りは PdfFileWriterオブジェクト を介して行われることになります。(<3.1 PdfFileMerge>も参考にして下さい)

編集には、「**構成要素を追加する**」「**ページを追加(挿入)・複製**」「**要素を削除する**」などの操作が行えます。

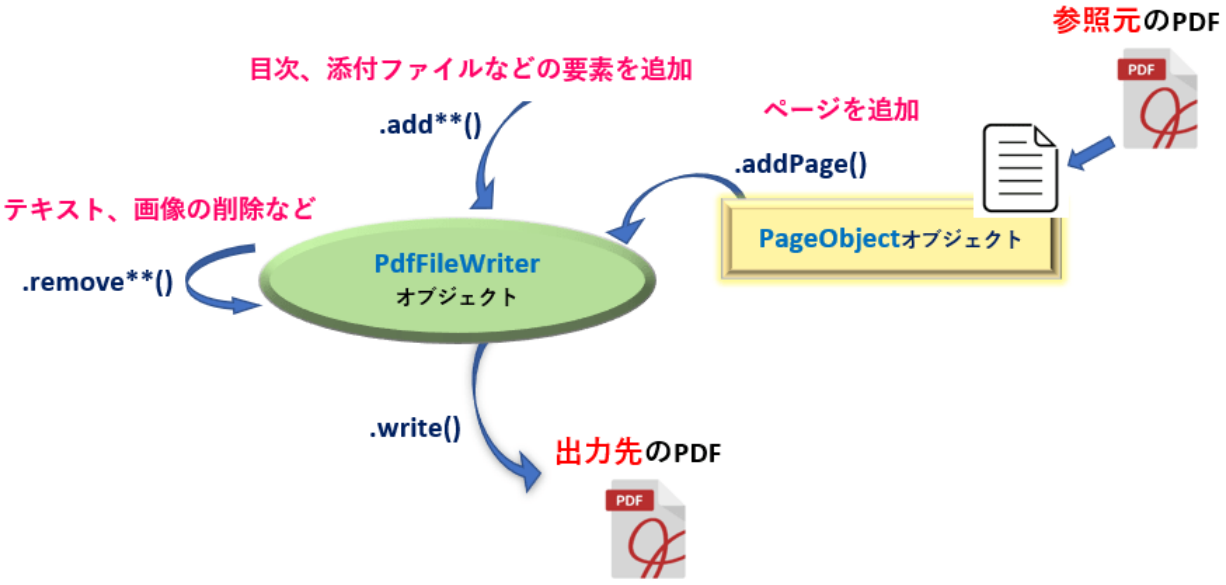


図12. PdfFileWriterオブジェクトの扱い

・　・　・

それでは、機能別にメソッドを紹介します。(主要なものに限定)

「**①.ファイルの構成要素を追加する**」

PDFに他のファイルを添付する、目次、パスワード、ファイル内リンクを設定するなど、構成要素の追加に関するメソッドがいくつか用意されています。

ファイルの構成要素を「追加」するメソッド

<ファイルを添付する(組み込む)>

PdfFileWriterオブジェクト.addAttachment(fname, fdata)

引数: fname：添付場所に表示させる名前を設定 (文字列)
引数: fdata：添付ファイルを設定 (文字列)

<目次(しおり)に追加する>

PdfFileWriterオブジェクト.addBookmark(title, pagenum, parent)

引数: title :目次に登録するタイトルを設定 (文字列)
引数: pagenum :目次を設定するページ番号 (整数)
引数: parent :親階層の目次を設定
(ReaderオブジェクトのgetOutlines()メソッドで取得できるDestinationsオブジェクトを設定)

<パスワードを設定する>

PdfFileWriterオブジェクト.encrypt(user_pwd)

引数: user_pwd **:**パスワードを設定する (文字列)

「②**.ページを追加(挿入)・複製する**」

PDFにページを追加するメソッドもいくつか用意されています。
追加するページ位置は、**最終ページ以降に追記するタイプ**や、**任意のページ以降に挿入するタイプ**があります。
また、設定するコンテンツには、**空白(ブランク)ページ**や、**参照ファイルから取得したページ情報**(PageObjectオブジェクト)を指定できます。
前者が `addBlankPage()メソッド` など、後者が `addPage()メソッド` などで以下の書式のようにして使います。

ページを追加する/挿入する

<ページを追加する (最後のページ以降)>

PdfFileWriterオブジェクト.addPage(Pageオブジェクト)

引数: page**:**追加するPageObjectオブジェクトを指定する

<ページを挿入する (任意のページ位置)>

PdfFileWriterオブジェクト.insertpage(page, index)

引数: page**:** 挿入するPageObjectオブジェクトを指定する
引数: index**:** 挿入するページを指定する(1ページ目が0[デフォルト])

<空白ページを追加する (最後のページ以降)>

PdfFileWriterオブジェクト.addBlankPage(width, height)

引数: width**:** 追加するページの横幅を指定
引数: height**:** 追加するページの縦幅を指定
戻り値: `PageObjectオブジェクト`

上記の書式では、個別ページ情報を1ページづつ追加(挿入)していきますが、ファイルの全ページをPdfFileReaderオブジェクト<リンク>として一度に追加(複製)することもできます。たとえば、次の`appendPagesFromReader()メソッド` を使うことで可能となります。(図13)

PdfFileReaderオブジェクトごと複製する

PdfFileWriterオブジェクト.appendPagesFromReader(reader)

引数: reader**:** 追加するPdfFileReaderオブジェクトを設定する

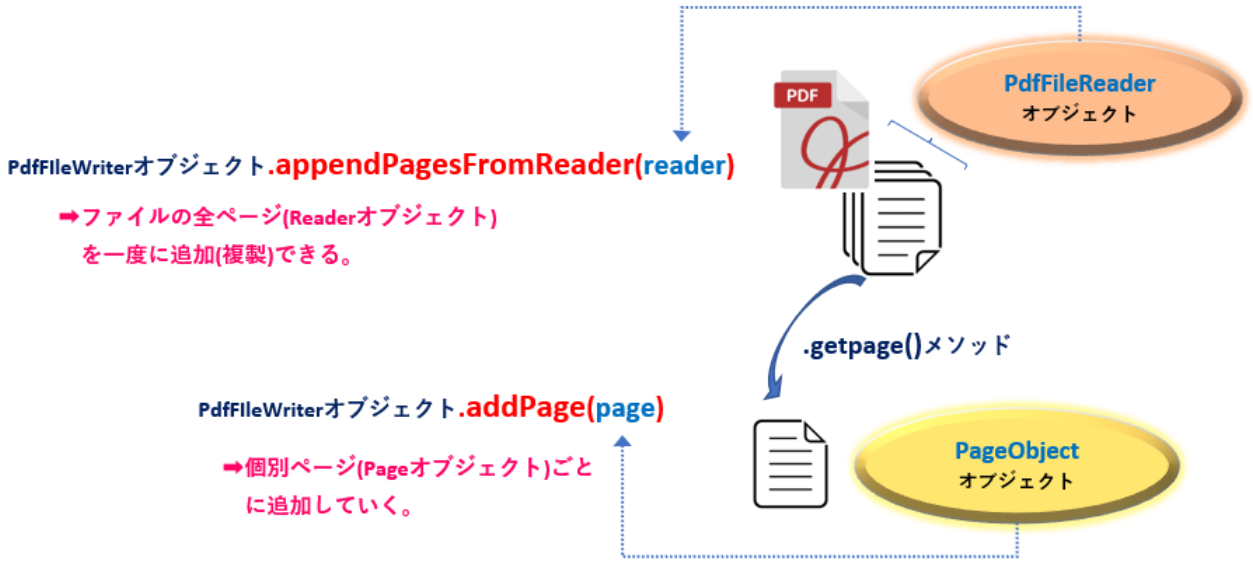


図13. ページの追加とコンテンツの指定

「③**. 構成要素を削除する**」

ここまでページや目次などPDFの構成要素を追加編集するメソッドを紹介しましたが、逆にそれらの要素を削除することも可能です。
たとえば、テキスト・画像などを削除する場合には次のようなメソッドがあります。

ファイルの構成要素を「削除」するメソッド

<テキストを削除する>

PdfFileWriterオブジェクト.removeText()

<画像を削除する>

PdfFileWriterオブジェクト.removeImages()

<リンクを削除する>

PdfFileWriterオブジェクト.removeLinks()

・　・　・

最後に、PdfFileWriterオブジェクトの内容を、出力先のファイルオブジェクト(“**bw**”バイナリ書き込みモード)へ **write**()メソッド にて書き出します。

Fileオブジェクトに書き出す

PdfFileWriterオブジェクト.write(stream)

引数: stream**:** 出力先のFileオブジェクトを指定する

PdfFileWriterクラスの主要メソッドの紹介でした。次項で具体的な活用例を確認しましょう。

4.2 PDFファイルをページ毎に分割するサンプルコード

PdfFileWriterクラスを使ったサンプルコードで具体的な使い方を確認してみましょう。

紹介するコードの概要は、既存の複数ページからなるＰＤＦファイルを1ページずつ分割して、独立したPDFファイルとしてパスワード付きで保存するというものです。

List4

```
from PyPDF2 import PdfFileReader # PdfFileReaderクラスのインポート
from PyPDF2 import PdfFileWriter # PdfFileWriterクラスのインポート

fp = open("PDF_pages.pdf", 'rb')
input = PdfFileReader(fp)

pgnum = input.getNumPages()

for i in range(pgnum):

    output = PdfFileWriter()                        # PdfFileWriterオブジェクトの取得
    output.addPage(input.getPage(i))                # PdfFileWriterオブジェクトにPageオブジェクトを追加
    output.encrypt("password")                      # パスワードを設定
    outputfile = open("pdf_"+str(i)+".pdf", 'wb')   # 新規Fileオブジェクトを'wb'モードで取得
    output.write(outputfile)                        # PdfFileWriterオブジェクトを新規Fileオブジェクトに書き出し
    outputfile.close()

fp.close()
```

それでは、コードのポイントを解説します。

冒頭で、 **PdfFileWriter**クラス と **PdfFileReader**クラス をインポートします。
4,5行目で編集元のPDFのファイルオブジェクトと **Reader**オブジェクト を取得します。
For文によりページ数分、以降の処理を繰り返します。

9~13行目：【PdfFileWriterオブジェクトの編集】

まず、11行目でページ毎ごとに新しいPdfFileWriterオブジェクトを取得します。
次に、12行目でPdfFileReaderオブジェクトの `getPage()メソッド` で読込むPDFファイルから個別ページのPageObjectオブジェクトを取得します。
そして、それを `addPage()メソッド` でPdfFileWriterオブジェクトに追記、パスワードを付与していきます。

14行目：【ファイルオブジェクトへ書き出し】

最後に、PdfFileWriterオブジェクトの内容をwrite()メソッドで出力先のFileオブジェクト(“**wb**”モード)に書き出します。

開いたファイルは閉じてからコードを終了します。

・　・　・

<List4>を実行した結果は次のようになりました。
4ページ分のPDFファイルが新たに作成されパスワード付きで保存されました。

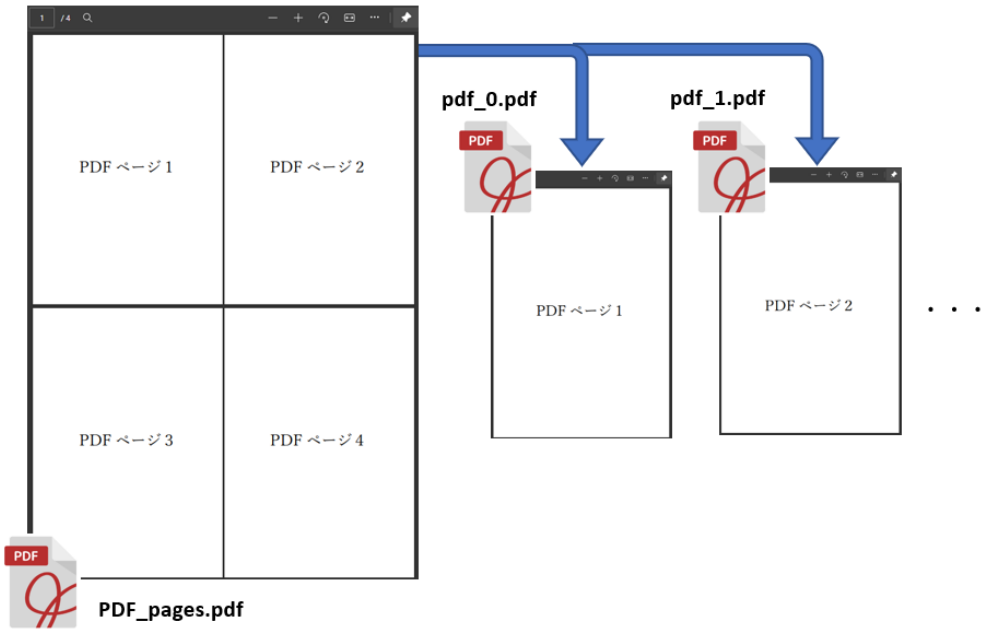
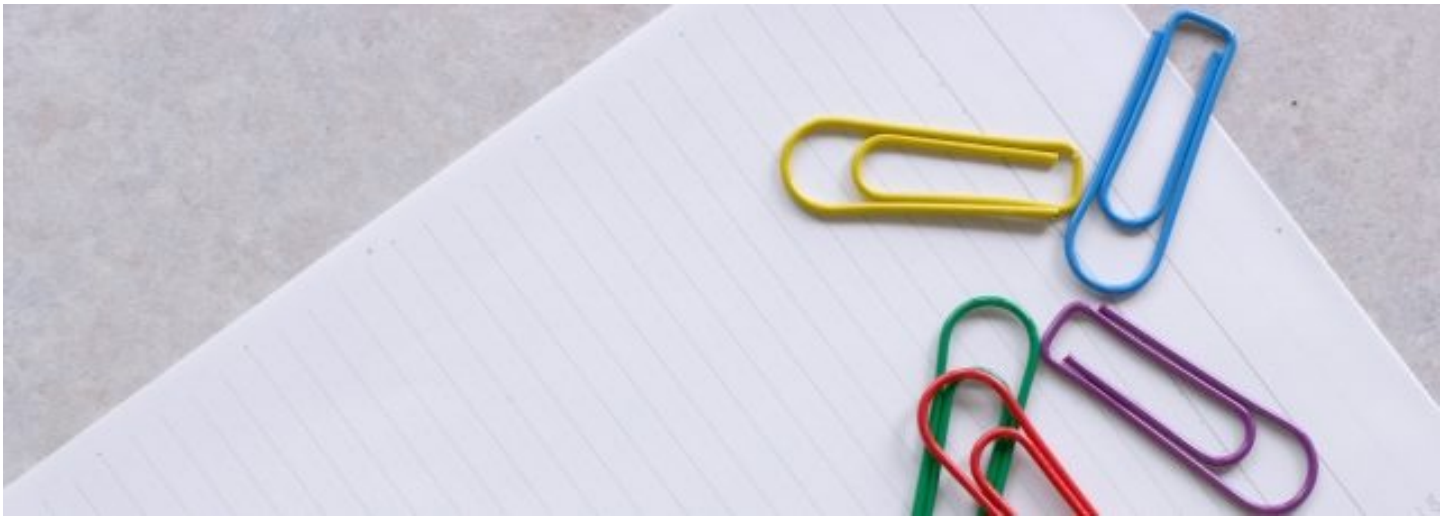


図15. <List 4 >の実行結果

5. まとめ



いかがでしたでしょうか？

今回はPDFファイルを操作するライブラリとしてPyPDF2を解説しました。
このライブラリを使うことで一度に大量のファイルを処理する必要がある場合やルーティン化したPDFに関連した作業をPythonで一括自動処理できます。

今回の記事がぜひ、あなたの仕事の効率アップにお役立てれば幸いです。

記事内容のポイントをまとめておきましょう。

✔ CHECK

①. **PDFファイルの作成・編集をすることができる。**（但し、ゼロからコンテンツをつくるのではなく既存のPDFをベースに作り込む。）

②. PyPDF2には主に「PdfFileReader」「PdfFileMerger」「PdfFileWriter」「PageObject」**の4つのクラスがある。それぞれPDFファイルの「読み込み」、「ファイルの結合」、「ファイルの編集」「コンテンツ抽出」などの機能を提供する。**

③. **日本語のテキストを抽出するには「PDFMiner」、ゼロからコンテンツを作るには「ReportLab」など他のライブラリとの併用を勧める。**

参考記事

【Python×PDF】PDFMinerライブラリでPDFからテキストを抽出方法【徹底解説】

この記事では「PDFMiner」ライブラリで、PDFファイルからテキスト(文章)コンテンツを抽出する方法を解説しています。ライブラリの紹介からインストール方法、実践まで参考になります。

www.shibutan-bloomers.com 2021.03.23

また、本記事は**日経BP社「日経ソフトウェア」**さんの**2020年3月号**の一部記事を参考にした上で、筆者独自の視点による解説を加えています。ぜひ、こちらの書籍も参考にスキル向上にお役立てください。

リンク

最後までお読みいただきありがとうございました。