How to Work With a PDF in Python

The Portable Document Format or PDF is a file format that can be used to present and exchange documents reliably across operating systems. While the PDF was originally invented by Adobe, it is now an open standard that is maintained by the International Organization for Standardization (ISO). You can work with a preexisting PDF in Python by using the **`PyPDF2`** package.

`PyPDF2` is a pure-Python package that you can use for many different types of PDF operations.

By the end of this article, you'll know how to do the following:

- Extract document information from a PDF in Python
- Rotate pages
- Merge PDFs
- Split PDFs

- Add watermarks
- Encrypt a PDF

History of `pyPdf`, `PyPDF2`, and `PyPDF4`

The original `pyPdf` package was released way back in 2005. The last official release of `pyPdf` was in 2010. After a lapse of around a year, a company called Phasit sponsored a fork of `pyPdf` called `PyPDF2`. The code was written to be backwards compatible with the original and worked quite well for several years, with its last release being in 2016.

There was a brief series of releases of a package called `PyPDF3`, and then the project was renamed to `PyPDF4`. All of these projects do pretty much the same thing, but the biggest difference between `pyPdf`and PyPDF2+ is that the latter versions added Python 3 support. There is a different Python 3 fork of the original `pyPdf` for Python 3, but that one has not been maintained for many years.

While PyPDF2 was recently abandoned, the new PyPDF4 does not have full backwards compatibility with PyPDF2. Most of the examples in this article will work perfectly fine with PyPDF4, but there are some that cannot, which is why PyPDF4 is not featured more heavily in this article. Feel free to swap out the imports for PyPDF2 with PyPDF4 and see how it works for you.

pdfrw: An Alternative

Patrick Maupin created a package called pdfrw that can do many of the same things that PyPDF2 does. You can use pdfrw for all of the same sorts of tasks that you will learn how to do in this article for PyPDF2, with the notable exception of encryption.

The biggest difference when it comes to pdfrw is that it integrates with the ReportLab package so that you can take a preexisting PDF and build a new one with ReportLab using some or all of the preexisting PDF.

Installation

Installing `PyPDF2` can be done with `pip` or `conda` if you happen to be using Anaconda instead of regular Python.

Here's how you would install `PyPDF2` with `pip`:

```
$ pip install pypdf2
```

The install is quite quick as `PyPDF2` does not have any dependencies. You will likely spend as much time downloading the package as you will installing it.

Now let's move on and learn how to extract some information from a PDF.

How to Extract Document Information From a PDF in Python

You can use `PyPDF2` to extract metadata and some text from a PDF. This can be useful when you're doing certain types of automation on your preexisting PDF files.

Here are the current types of data that can be extracted:

- Author
- Creator
- Producer
- Subject
- Title
- Number of pages

You need to go find a PDF to use for this example. You can use any PDF you have handy on your machine. To make things easy, I went to Leanpub and grabbed a sample of one of my books for this exercise. The sample you want to download is called `reportlab-sample.pdf`.

Let's write some code using that PDF and learn how you can get access to these attributes:

```
# extract_doc_info.py

from PyPDF2 import PdfFileReader

def extract_information(pdf_path):
    with open(pdf_path, 'rb') as f:
        pdf = PdfFileReader(f)
        information = pdf.getDocumentInfo(
        number_of_pages = pdf.getNumPages(

    txt = f"""
    Information about {pdf_path}:

    Author: {information.author}
    Creator: {information.creator}
    Producer: {information.producer}
    Subject: {information.subject}
    Title: {information.title}
    Number of pages: {number_of_pages}
    """

    print(txt)
    return information

if __name__ == '__main__':
```

```
    path = 'reportlab-sample.pdf'
    extract_information(path)
```

Here you import `PdfFileReader` from the `PyPDF2` package. The `PdfFileReader` is a class with several methods for interacting with PDF files. In this example, you call `.getDocumentInfo()`, which will return an instance of `DocumentInformation`. This contains most of the information that you're interested in. You also call `.getNumPages()` on the reader object, which returns the number of pages in the document.

> **Note:** That last code block uses Python 3's new f-strings for string formatting. If you'd like to learn more, you can check out Python 3's f-Strings: An Improved String Formatting Syntax (Guide).

The `information` variable has several instance attributes that you can use to get the rest of the metadata you want from the document. You print out that information and also return it for potential future use.

While `PyPDF2` has `.extractText()`, which can be used on its page objects (not shown in this example), it does not work very well. Some PDFs will return text and some will return an empty string. When you want to extract text from a PDF, you should check out the `PDFMiner` project instead. **`PDFMiner`** is much more robust and was specifically designed for extracting text from PDFs.

Now you're ready to learn about rotating PDF pages.

How to Rotate Pages

Occasionally, you will receive PDFs that contain pages that are in landscape mode instead of portrait mode. Or perhaps they are even upside down. This can happen when someone scans a document to PDF or email. You could print the document out and read the paper version or you can use the power of Python to rotate the offending pages.

For this example, you can go and pick out a Real Python article and print it to PDF.

Let's learn how to rotate a few of the pages of that article with `PyPDF2`:

```
# rotate_pages.py

from PyPDF2 import PdfFileReader, PdfFileW

def rotate_pages(pdf_path):
    pdf_writer = PdfFileWriter()
    pdf_reader = PdfFileReader(pdf_path)
    # Rotate page 90 degrees to the right
    page_1 = pdf_reader.getPage(0).rotate(
    pdf_writer.addPage(page_1)
    # Rotate page 90 degrees to the left
    page_2 = pdf_reader.getPage(1).rotate(
    pdf_writer.addPage(page_2)
    # Add a page in normal orientation
    pdf_writer.addPage(pdf_reader.getPage(

    with open('rotate_pages.pdf', 'wb') as
        pdf_writer.write(fh)

if __name__ == '__main__':
    path = 'Jupyter_Notebook_An_Introducti
    rotate_pages(path)
```

For this example, you need to import the `PdfFileWriter` in addition to `PdfFileReader` because you will need to write out a new PDF. `rotate_pages()` takes in the path to the PDF that you want to modify. Within that function, you will need to create a writer object that you can name `pdf_writer` and a reader object called `pdf_reader`.

Next, you can use `.GetPage()` to get the desired page. Here you grab page zero, which is the first page. Then you call the page object's `.rotateClockwise()` method and pass in 90 degrees. Then for page two, you call `.rotateCounterClockwise()` and pass it 90 degrees as well.

> **Note:** The `PyPDF2` package only allows you to rotate a page in increments of 90 degrees. You will receive an `AssertionError` otherwise.

After each call to the rotation methods, you call `.addPage()`. This will add the rotated version of the page to the writer object. The last page

that you add to the writer object is page 3 without any rotation done to it.

Finally you write out the new PDF using `.write()`. It takes a file-like object as its parameter. This new PDF will contain three pages. The first two will be rotated in opposite directions of each other and be in landscape while the third page is a normal page.

Now let's learn how you can merge multiple PDFs into one.

How to Merge PDFs

There are many situations where you will want to take two or more PDFs and merge them together into a single PDF. For example, you might have a standard cover page that needs to go on to many types of reports. You can use Python to help you do that sort of thing.

For this example, you can open up a PDF and print a page out as a separate PDF. Then do that again, but with a different page. That will give

you a couple of inputs to use for example purposes.

Let's go ahead and write some code that you can use to merge PDFs together:

```
# pdf_merging.py

from PyPDF2 import PdfFileReader, PdfFileW

def merge_pdfs(paths, output):
    pdf_writer = PdfFileWriter()

    for path in paths:
        pdf_reader = PdfFileReader(path)
        for page in range(pdf_reader.getNu
            # Add each page to the writer
            pdf_writer.addPage(pdf_reader.

    # Write out the merged PDF
    with open(output, 'wb') as out:
        pdf_writer.write(out)

if __name__ == '__main__':
    paths = ['document1.pdf', 'document2.p
    merge_pdfs(paths, output='merged.pdf')
```

You can use `merge_pdfs()` when you have a list of PDFs that you want to merge together. You will also need to know where to save the result,

so this function takes a list of input paths and an output path.

Then you loop over the inputs and create a PDF reader object for each of them. Next you will iterate over all the pages in the PDF file and use `.addPage()` to add each of those pages to itself.

Once you're finished iterating over all of the pages of all of the PDFs in your list, you will write out the result at the end.

One item I would like to point out is that you could enhance this script a bit by adding in a range of pages to be added if you didn't want to merge all the pages of each PDF. If you'd like a challenge, you could also create a command line interface for this function using Python's `argparse` module.

Let's find out how to do the opposite of merging!

How to Split PDFs

There are times where you might have a PDF that you need to split up into multiple PDFs. This is especially true of PDFs that contain a lot of scanned-in content, but there are a plethora of good reasons for wanting to split a PDF.

Here's how you can use `PyPDF2` to split your PDF into multiple files:

```
# pdf_splitting.py

from PyPDF2 import PdfFileReader, PdfFileW

def split(path, name_of_split):
    pdf = PdfFileReader(path)
    for page in range(pdf.getNumPages()):
        pdf_writer = PdfFileWriter()
        pdf_writer.addPage(pdf.getPage(pag

        output = f'{name_of_split}{page}.p
        with open(output, 'wb') as output_
            pdf_writer.write(output_pdf)

if __name__ == '__main__':
    path = 'Jupyter_Notebook_An_Introducti
    split(path, 'jupyter_page')
```

In this example, you once again create a PDF reader object and loop over its pages. For each page in the PDF, you will create a new PDF writer instance and add a single page to it. Then you will write that page out to a uniquely named

file. When the script is finished running, you should have each page of the original PDF split into separate PDFs.

Now let's take a moment to learn how you can add a watermark to your PDF.

How to Add Watermarks

Watermarks are identifying images or patterns on printed and digital documents. Some watermarks can only be seen in special lighting conditions. The reason watermarking is important is that it allows you to protect your intellectual property, such as your images or PDFs. Another term for watermark is overlay.

You can use Python and `PyPDF2` to watermark your documents. You need to have a PDF that only contains your watermark image or text.

Let's learn how to add a watermark now:

```
# pdf_watermarker.py

from PyPDF2 import PdfFileWriter, PdfFileR

def create_watermark(input_pdf, output, wa
    watermark_obj = PdfFileReader(waterma
    watermark_page = watermark_obj.getPage

    pdf_reader = PdfFileReader(input_pdf)
    pdf_writer = PdfFileWriter()

    # Watermark all the pages
    for page in range(pdf_reader.getNumPag
        page = pdf_reader.getPage(page)
        page.mergePage(watermark_page)
        pdf_writer.addPage(page)

    with open(output, 'wb') as out:
        pdf_writer.write(out)

if __name__ == '__main__':
    create_watermark(
        input_pdf='Jupyter_Notebook_An_Int
        output='watermarked_notebook.pdf',
        watermark='watermark.pdf')
```

`create_watermark()` accepts three arguments:

1. **`input_pdf:`** the PDF file path to be watermarked
2. **`output:`** the path you want to save the watermarked version of the PDF
3. **`watermark:`** a PDF that contains your watermark image or text

In the code, you open up the watermark PDF and grab just the first page from the document as that is where your watermark should reside. Then you create a PDF reader object using the `input_pdf` and a generic `pdf_writer` object for writing out the watermarked PDF.

The next step is to iterate over the pages in the `input_pdf`. This is where the magic happens. You will need to call `.mergePage()` and pass it the `watermark_page`. When you do that, it will overlay the `watermark_page` on top of the current page. Then you add that newly merged page to your `pdf_writer`object.

Finally, you write the newly watermarked PDF out to disk, and you're done!

The last topic you will learn about is how PyPDF2 handles encryption.

How to Encrypt a PDF

PyPDF2 currently only supports adding a user password and an owner password to a preexisting PDF. In PDF land, an owner password will basically give you administrator privileges over the PDF and allow you to set permissions on the document. On the other hand, the user password just allows you to open the document.

As far as I can tell, PyPDF2 doesn't actually allow you to set any permissions on the document even though it does allow you to set the owner password.

Regardless, this is how you can add a password, which will also inherently encrypt the PDF:

```
# pdf_encrypt.py

from PyPDF2 import PdfFileWriter, PdfFileR

def add_encryption(input_pdf, output_pdf,
    pdf_writer = PdfFileWriter()
    pdf_reader = PdfFileReader(input_pdf)

    for page in range(pdf_reader.getNumPag
        pdf_writer.addPage(pdf_reader.getP

    pdf_writer.encrypt(user_pwd=password,
                       use_128bit=True)

    with open(output_pdf, 'wb') as fh:
        pdf_writer.write(fh)

if __name__ == '__main__':
    add_encryption(input_pdf='reportlab-sa
                   output_pdf='reportlab-e
                   password='twofish')
```

`add_encryption()` takes in the input and output PDF paths as well as the password that you want to add to the PDF. It then opens a PDF

writer and a reader object, as before. Since you will want to encrypt the entire input PDF, you will need to loop over all of its pages and add them to the writer.

The final step is to call `.encrypt()`, which takes the user password, the owner password, and whether or not 128-bit encryption should be added. The default is for 128-bit encryption to be turned on. If you set it to `False`, then 40-bit encryption will be applied instead.

> **Note:** PDF encryption uses either RC4 or AES (Advanced Encryption Standard) to encrypt the PDF according to pdflib.com.
>
> Just because you have encrypted your PDF does not mean it is necessarily secure. There are tools to remove passwords from PDFs. If you'd like to learn more, Carnegie Mellon University has an interesting paper on the topic.

Conclusion

The `PyPDF2` package is quite useful and is usually pretty fast. You can use `PyPDF2` to automate large jobs and leverage its capabilities to help you do your job better!

In this tutorial, you learned how to do the following:

- Extract metadata from a PDF
- Rotate pages
- Merge and split PDFs
- Add watermarks
- Add encryption

Also keep an eye on the newer `PyPDF4` package as it will likely replace `PyPDF2` soon. You might also want to check out `pdfrw`, which can do many of the same things that `PyPDF2` can do.