Chapter 1: Introduction to PDF Processing in Python

Welcome to the first step of your journey in Python-based PDF processing! This chapter will introduce you to PyMuPDF, a versatile Python library used for a wide array of PDF manipulation tasks.

In this chapter, we'll get you acquainted with the PyMuPDF library, which presents a user-friendly Pythonic interface for handling PDF files. We'll cover the installation process, and then dive straight into reading PDF documents, navigating through their structure, loading pages, and extracting text.

But we won't stop at reading PDFs. PyMuPDF is also powerful in manipulating PDF documents, so we'll walk you through splitting and merging PDF documents, essential skills in many PDF processing tasks.

By the end of this chapter, you'll be well-versed in the basics of handling PDFs using PyMuPDF, setting a solid foundation for more advanced topics in the following chapters. So let's get started!

Reading PDF Files

Getting Started

For the first section of this book, we're going to learn how to read a PDF file using Python and the PyMuPDF library. Reading PDF files is a fundamental skill and it's the first step when you're working with this type of file format.

We'll start with the basics: opening a PDF file, navigating its pages, and extracting text. By the end of this section, you'll be able to read any PDF file with Python.

Installation of PyMuPDF

Before we start reading PDFs, let's install the PyMuPDF library. PyMuPDF is a powerful library for PDF processing that provides a wide range of features, including text extraction, image extraction, link extraction, and many more.

To install it, open your terminal or command prompt and type the following command:

```
$ pip install PyMuPDF
```

This command tells `pip`, Python's package installer, to download and install the PyMuPDF library from the Python Package Index (PyPI). If the installation is successful, you should see a message saying that PyMuPDF was successfully installed.

Opening a PDF File

With PyMuPDF installed, we can now start working with PDF documents. The first step to reading a PDF file is to load it, or open it. We do this using the `fitz.open()` function, which takes the path to the PDF file as the argument:

```
import fitz  # PyMuPDF

# Open the PDF
doc = fitz.open('Chapter_2_Building_Malware.pdf')
```

The `fitz.open()` function returns a `fitz.fitz.Document` object that represents the PDF file. We'll use this object to interact with the PDF file'.

The `Chapter_2_Building_Malware.pdf` is our free chapter that is part of the Ethical Hacking with Python EBook, you can get this sample PDF document here.

Navigating the Document

Once we have a `Document` object, we can start navigating through the PDF file. For example, let's find out how many pages the document has using the `page_count` attribute:

```
# Get the number of pages
print("Number of pages: ", doc.page_count)
```

This command will print the number of pages in our PDF document. Here's the output:

```
Number of pages:  60
```

Loading a Page

To read the content of a specific page, we first need to load it. We do this using the `load_page()` method of the `Document` object:

```
# load the first page
first_page = doc.load_page(0)  # 0 represents the first page
```

This `load_page()` method returns a `fitz.fitz.Page` object that represents the loaded page. The page numbers start from 0 and not 1. So `doc.load_page(0)` should load the first page of the document, `doc.load_page(1)` loads the second page and so on.

The `Page` object has many properties and methods that allow us to interact with the content of the PDF page. In this section, we're particularly interested in the `get_text()` method, which allows us to extract the raw text from the page.

Extracting Text from a Page

Let's use the `get_text()` method to extract all text from the page:

```
# Get the text of the first page
first_page_text = first_page.get_text()
print(first_page_text)
```

This will print the string that contains all the text in the page. Here's a part of the output:

```
Chapter 2: Building Malware
Malware is a computer program designed to attack a computer system. Malware
is often used to steal data from a user's computer or damage a computer system.
In this chapter, we will learn how to build malware using Python. Below are the
programs we will be making:
...<SNIPPED>...
```

The output is quite long (as it's an entire page), so I'm just printing the first 5 lines.

Reading Multiple Pages

If we want to read all the pages in the document, we can use a simple `for` loop:

```
# Get the text of all the pages
for i in range(doc.page_count):
    # load the page
    page = doc.load_page(i)
    # get the text
    page_text = page.get_text()
    # print the text
    print(f"--- Page {i+1} ---")
    print(page_text)
```

This code block will print the text from each page, preceded by a header that indicates the page number, meaning it will print all the chapter text.

In PyMuPDF, we don't need to manually close the PDF file or the page, the library will automatically close it when the `Document` object is deleted, goes out of scope, or when the program exits.

Wrapping up the Code

In this subsection, we're going to make a command-line script that does everything we did above, but the code is wrapped in a function, and the `argparse` module is used to parse the command-line arguments passed by the user. Open up a new Python file named `extract_text_from_pdf.py` and add the following:

```
import fitz
import argparse
import os

# wrap the above in a function
def extract_text(pdf_file, output_file, by_page=False, output_folder="results"):
    """Extract the text from a PDF file and save it to a text file.
    If by_page is True, each page is saved to a separate text file.
    Args:
        pdf_file (str): Path to the PDF file to be extracted.
        output_file (str): Path to the output file.
        by_page (bool, optional): If True, save each page to a separate text
file.
            Defaults to False."""
    # create the output folder if it doesn't exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)
    # open the pdf
    doc = fitz.open(pdf_file)
    # get the number of pages
    num_pages = doc.page_count
    # if by_page is True, save each page to a separate text file
    if by_page:
        # get the name of the output file
        file_name = os.path.basename(output_file)
        # get the name of the output file without the extension
        file_name = os.path.splitext(file_name)[0]
        # iterate over the pages
        for i in range(num_pages):
            # load the page
            page = doc.load_page(i)
            # get the text
            page_text = page.get_text()
            # create a text file
            with open(os.path.join(output_folder, f"{file_name}_page_{i+1}.txt"),
"w") as f:
                # write the text to the file
                f.write(page_text)
    else:
        # get the text of all the pages
        for i in range(num_pages):
            # load the page
            page = doc.load_page(i)
            # get the text
            page_text = page.get_text()
            # create a text file
            with open(os.path.join(output_folder, output_file), "a") as f:
                # add a header to separate the pages
                f.write(f"--- Page {i+1} ---\n")
                # write the text to the file
                f.write(page_text)
```

This function extracts the text from any PDF file and saves it to a text file. There are a total of four parameters on this function:

- `pdf_file`: The path of the target PDF document.
- `output_file`: The output text file name.
- `by_page`: A boolean indicating whether we want to save each page to a separate text file.
- `output_folder`: The target folder where we will save our output text file(s).

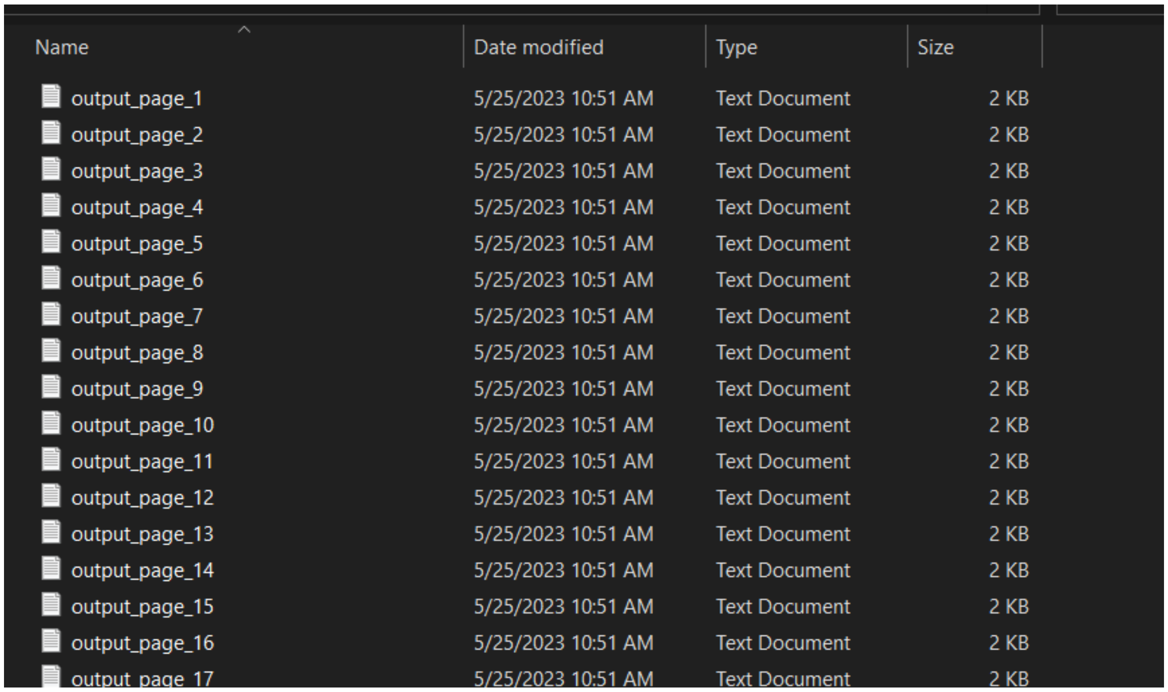
Let's use the `argparse` module now to parse the passed arguments and pass them to the `extract_text()` function:

```
if __name__ == "__main__":
    # create the parser object
    parser = argparse.ArgumentParser(description="Extract the text from a PDF
file.")
    # add the arguments
    parser.add_argument("pdf_file", help="Path to the PDF file to be extracted.")
    parser.add_argument("output_file", help="Path to the output file.")
    parser.add_argument("--output_folder", help="Path to the output folder.",
default="results")
    parser.add_argument(
        "--by_page",
        help="If True, save each page to a separate text file.",
        action="store_true",
    )
    # parse the arguments
    args = parser.parse_args()
    # extract the text
    extract_text(args.pdf_file, args.output_file, args.by_page)
```

Excellent! Let's use the script now:

```
$ python extract_text_from_pdf.py Chapter_2_Building_Malware.pdf output.txt
--by_page --output_folder results
```

This will create a folder named `results` that contain all the text of all the pages of our PDF file:



Name	Date modified	Type	Size
output_page_1	5/25/2023 10:51 AM	Text Document	2 KB
output_page_2	5/25/2023 10:51 AM	Text Document	2 KB
output_page_3	5/25/2023 10:51 AM	Text Document	2 KB
output_page_4	5/25/2023 10:51 AM	Text Document	2 KB
output_page_5	5/25/2023 10:51 AM	Text Document	2 KB
output_page_6	5/25/2023 10:51 AM	Text Document	2 KB
output_page_7	5/25/2023 10:51 AM	Text Document	2 KB
output_page_8	5/25/2023 10:51 AM	Text Document	2 KB
output_page_9	5/25/2023 10:51 AM	Text Document	2 KB
output_page_10	5/25/2023 10:51 AM	Text Document	2 KB
output_page_11	5/25/2023 10:51 AM	Text Document	2 KB
output_page_12	5/25/2023 10:51 AM	Text Document	2 KB
output_page_13	5/25/2023 10:51 AM	Text Document	2 KB
output_page_14	5/25/2023 10:51 AM	Text Document	2 KB
output_page_15	5/25/2023 10:51 AM	Text Document	2 KB
output_page_16	5/25/2023 10:51 AM	Text Document	2 KB
output_page_17	5/25/2023 10:51 AM	Text Document	2 KB

If you want to get them into a single output file, simply remove the `--by_page` argument:

```
$ python extract_text_from_pdf.py Chapter_2_Building_Malware.pdf output.txt
--output_folder results
```

Congratulations! You've learned the first step towards mastering PDF Files handling with Python, and that's reading and extracting text from PDFs with PyMuPDF. You can now open a PDF file, navigate through its pages, and extract the text from each page. This is a fundamental skill when working with PDF files in Python, and it's the foundation for more advanced tasks.

Splitting PDF Files

Getting Started

Splitting PDF documents can be useful when you need to separate individual sections of a document for independent distribution or analysis. In this section, we will explore how to split PDF files into separate documents, by individual page numbers, or ranges with the help of the PyMuPDF library in Python.

If you haven't installed PyMuPDF, make sure to do so:

```
$ pip install PyMuPDF
```

We will be building three individual Python scripts:

- `split_pdf_simple.py`: For splitting PDF documents into separate files, where each page of the original document is an individual file.
- `split_pdf_pages.py`: This one splits PDFs with a list of individual page numbers. For example, see the following list of lists:
  ```
  # set of page numbers, each element is a list of individual page numbers
  file_pages = [
      [0, 5, 6, 10],
      [12, 33, 50],
      [3, 9, 11, 14, 27, 39, 43, 51, 58],
  ]
  ```
  This will generate a total of 3 PDFs, the first one contains the page 0, 5, 6, and 10 of the original document, the second contains the page 12, 33, and 50, and so on.
- `split_pdf_ranges.py`: This script splits PDF documents with page number ranges. We will pass a Python list where each element is a tuple of (`start_page`, `end_page`) for each file.

Splitting By Individual Pages

Without further ado, let's get started with the simplest code, `split_pdf_simple.py`:

```
import fitz
import sys
import os
# get the filename
pdf_filename = sys.argv[1]
# output folder
output_folder = "pages" # change this to your desired folder
```

```python
# make the folder if it doesn't exist
if not os.path.exists(output_folder):
    os.makedirs(output_folder)
# load the document
doc = fitz.open(pdf_filename)
for i in range(len(doc)):
    # make a new document
    new_doc = fitz.open()
    # insert the page
    new_doc.insert_pdf(doc, from_page=i, to_page=i)
    # save the document with a meaningful name
    new_doc.save(os.path.join(output_folder, f"{pdf_filename[:-4]}_page{i}.pdf"))
```

First, we use the `sys` module to get the target PDF file from the command-line, then we make a folder named `pages` to save our resulting PDF files there.

After that, we open our PDF file we want to split with the `fitz.open()` function, and iterate over each page index in the document with `for i in range(len(doc))`. For each page, we create a new `Document` object with `fitz.open()`.

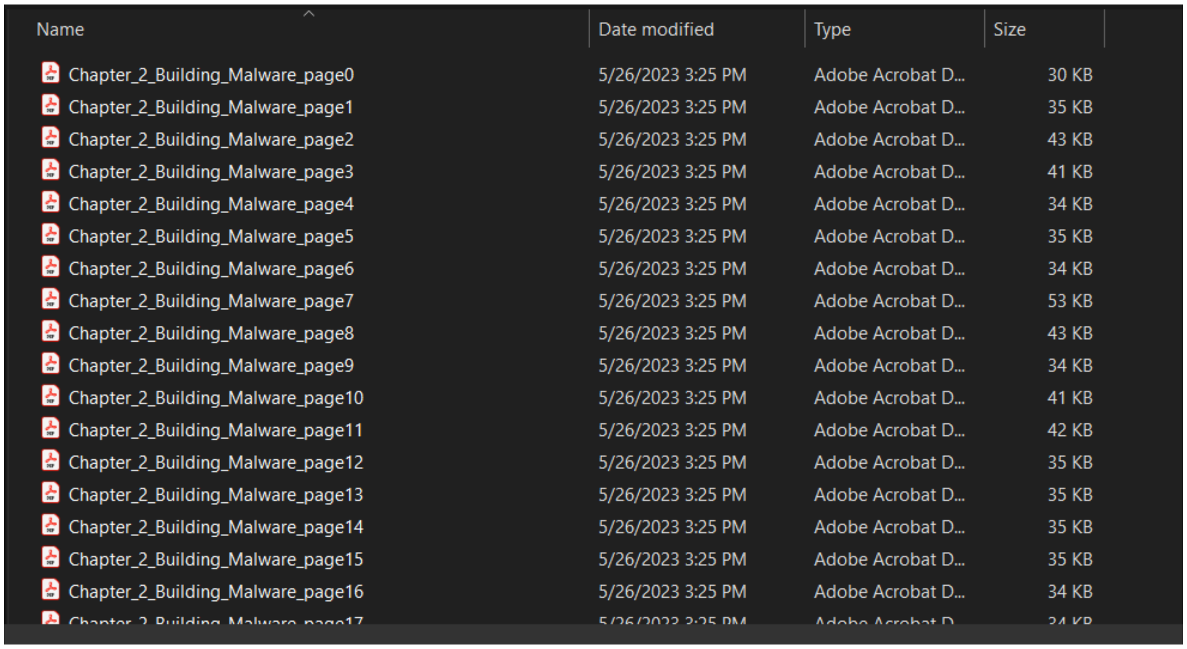
Then, we insert the current page from the original document into the new document using `insert_pdf()` method. The `from_page` and `to_page` parameters specify the range of pages to insert. In this case, we insert a single page so they are the same (`i`).

Finally, we save the new document with `new_doc.save()`. We use an f-string to generate a unique filename for that new PDF file based on the original document's name and the page number.

Let's run it:

```
$ python split_pdf_simple.py Chapter_2_Building_Malware.pdf
```

There is no output on the screen (by the way, you can add some logging, for example that particular PDF file is saved, etc.). However, notice the `pages` folder is created and contain all the pages of our PDF file as separate PDF files:



Name	Date modified	Type	Size
Chapter_2_Building_Malware_page0	5/26/2023 3:25 PM	Adobe Acrobat D...	30 KB
Chapter_2_Building_Malware_page1	5/26/2023 3:25 PM	Adobe Acrobat D...	35 KB
Chapter_2_Building_Malware_page2	5/26/2023 3:25 PM	Adobe Acrobat D...	43 KB
Chapter_2_Building_Malware_page3	5/26/2023 3:25 PM	Adobe Acrobat D...	41 KB
Chapter_2_Building_Malware_page4	5/26/2023 3:25 PM	Adobe Acrobat D...	34 KB
Chapter_2_Building_Malware_page5	5/26/2023 3:25 PM	Adobe Acrobat D...	35 KB
Chapter_2_Building_Malware_page6	5/26/2023 3:25 PM	Adobe Acrobat D...	34 KB
Chapter_2_Building_Malware_page7	5/26/2023 3:25 PM	Adobe Acrobat D...	53 KB
Chapter_2_Building_Malware_page8	5/26/2023 3:25 PM	Adobe Acrobat D...	43 KB
Chapter_2_Building_Malware_page9	5/26/2023 3:25 PM	Adobe Acrobat D...	34 KB
Chapter_2_Building_Malware_page10	5/26/2023 3:25 PM	Adobe Acrobat D...	41 KB
Chapter_2_Building_Malware_page11	5/26/2023 3:25 PM	Adobe Acrobat D...	42 KB
Chapter_2_Building_Malware_page12	5/26/2023 3:25 PM	Adobe Acrobat D...	35 KB
Chapter_2_Building_Malware_page13	5/26/2023 3:25 PM	Adobe Acrobat D...	35 KB
Chapter_2_Building_Malware_page14	5/26/2023 3:25 PM	Adobe Acrobat D...	35 KB
Chapter_2_Building_Malware_page15	5/26/2023 3:25 PM	Adobe Acrobat D...	35 KB
Chapter_2_Building_Malware_page16	5/26/2023 3:25 PM	Adobe Acrobat D...	34 KB
Chapter_2_Building_Malware_page17	5/26/2023 3:25 PM	Adobe Acrobat D	34 KB

Splitting by Arbitrary Page Groups

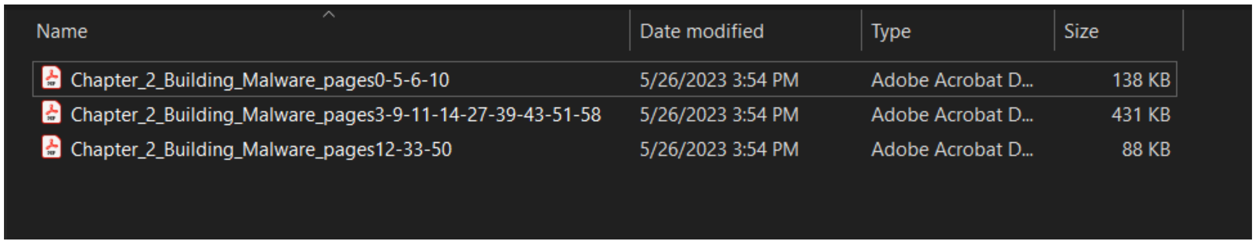
Let's now dive into the `split_pdf_pages.py`:

```python
import fitz
import sys
import os
# get the filename
pdf_filename = sys.argv[1]
# output folder
output_folder = "pages" # change this to your desired folder
# set of page numbers, each element is a list of individual page numbers
file_pages = [
    [0, 5, 6, 10],
    [12, 33, 50],
    [3, 9, 11, 14, 27, 39, 43, 51, 58],
]
# make the folder if it doesn't exist
if not os.path.exists(output_folder):
    os.makedirs(output_folder)
# load the document
doc = fitz.open(pdf_filename)
for file_page in file_pages:
    # make a new document
    new_doc = fitz.open()
    # insert the page
    for page in file_page:
        new_doc.insert_pdf(doc, from_page=page, to_page=page)
    # save the document with a meaningful name
    new_doc.save(os.path.join(output_folder,
f"{pdf_filename[:-4]}_pages{'-'.join([str(i) for i in file_page])}.pdf"))
```

This time, instead of iterating over all pages and creating a file for each page, we instead iterate over our handly constructed `file_pages` list, where each element is a list of individual page numbers. Let's run it:

```
$ python split_pdf_pages.py Chapter_2_Building_Malware.pdf
```

Here's how `pages` folder looks like:



Name	Date modified	Type	Size
Chapter_2_Building_Malware_pages0-5-6-10	5/26/2023 3:54 PM	Adobe Acrobat D...	138 KB
Chapter_2_Building_Malware_pages3-9-11-14-27-39-43-51-58	5/26/2023 3:54 PM	Adobe Acrobat D...	431 KB
Chapter_2_Building_Malware_pages12-33-50	5/26/2023 3:54 PM	Adobe Acrobat D...	88 KB

Excellent! This script is a more advanced version of the single-page splitting script we saw previously. It allows for arbitrary groups of pages to be included in each output file, rather than just one page per file.

Splitting by Page Ranges

Now in case you want to split by range, then it's simpler. Here's the code for `split_pdf_ranges.py`:

```
import fitz
import sys
import os
# get the filename
pdf_filename = sys.argv[1]
# output folder
output_folder = "pages" # change this to your desired folder
```

```
# set page numbers, each element is a tuple of (start_page, end_page) for each
file
file_pages = [(0, 10), (10, 18), (18, 25), (25, 59)]
# make the folder if it doesn't exist
if not os.path.exists(output_folder):
    os.makedirs(output_folder)
# load the document
doc = fitz.open(pdf_filename)
for file_page in file_pages:
    # make a new document
    new_doc = fitz.open()
    # insert the page
    new_doc.insert_pdf(doc, from_page=file_page[0], to_page=file_page[1])
    # save the document with a meaningful name
    new_doc.save(os.path.join(output_folder,
f"{pdf_filename[:-4]}_pages{'-'.join([str(i) for i in file_page])}.pdf"))
```

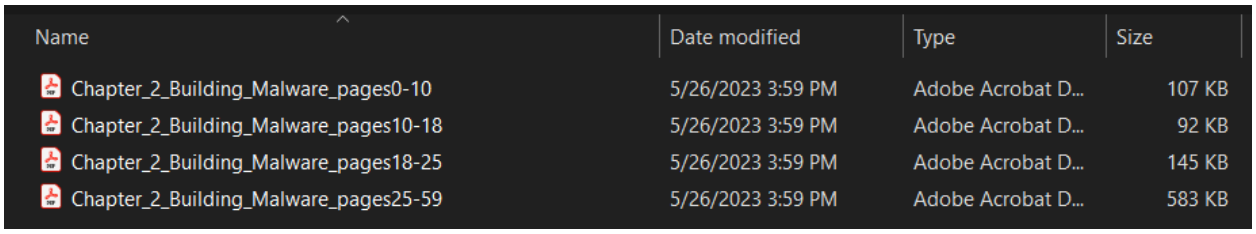
Now this is also similar to the previous ones, with the main difference being how it specifies which pages to include in each output file.

In this code, the page groups are defined as ranges rather than individual pages. `file_pages` is a list of tuples, where each tuple represents a range of pages (`start_page`, `end_page`) to be included in one output PDF file.

Let's run it:

```
$ python split_pdf_ranges.py Chapter_2_Building_Malware.pdf
```

Output files:



Name	Date modified	Type	Size
Chapter_2_Building_Malware_pages0-10	5/26/2023 3:59 PM	Adobe Acrobat D...	107 KB
Chapter_2_Building_Malware_pages10-18	5/26/2023 3:59 PM	Adobe Acrobat D...	92 KB
Chapter_2_Building_Malware_pages18-25	5/26/2023 3:59 PM	Adobe Acrobat D...	145 KB
Chapter_2_Building_Malware_pages25-59	5/26/2023 3:59 PM	Adobe Acrobat D...	583 KB

Conclusion

Great! Splitting PDF documents with Python and the PyMuPDF library is a powerful and flexible operation that has numerous practical applications. It allows you to manipulate large PDF documents with ease, dividing them into more manageable or specific parts based on individual pages, page ranges, or arbitrary groups of pages.

Whether you're looking to extract a single page, a range of pages, or non-consecutive pages, PyMuPDF has got you covered. It is easy to use and offers a clear and readable syntax. The library also includes methods for creating new documents and saving them with custom names, providing an end-to-end solution for splitting PDFs.

Merging PDF Files

Getting Started

Merging PDF documents is essential in scenarios where you need to combine different documents or pages into one PDF for easy distribution or management.

In this section, we're going to build a Python script that merges any number of PDF documents into one.

If you haven't already install PyMuPDF, please do:

```
$ pip install PyMuPDF
```

Parsing the Command-line Arguments

Open up a new Python file, name it `pdf_merger.py` and add the following:

```python
import fitz
import argparse
# make the parser
parser = argparse.ArgumentParser(description='Merge PDF files')
# add the input PDFs argument
parser.add_argument('files', nargs='+', help='PDF files to merge')
# parse the arguments
args = parser.parse_args()
```

We're using the argparse module so that we can accept any number of PDF files in the command-line. The `nargs='+'` options means one or more arguments of this type can be provided. These arguments will be accessible as a list with `args.files`.

Next, let's create a new PDF `Document` object with PyMuPDF:

```
# create the new output PDF file
output_file = fitz.open()
```

Performing the Merge

Now we iterate over the input PDF files and insert them to our `output_file`:

```
# loop through the PDFs and add them to the output file
for pdf_file in args.files:
    print(f'Adding {pdf_file}')
    input_file = fitz.open(pdf_file)
    output_file.insert_pdf(input_file)
```

For each PDF, we open it, and use the `output_file.insert_file()` method to add it to the new PDF `Document` object.

Finally, let's save the merged PDF:

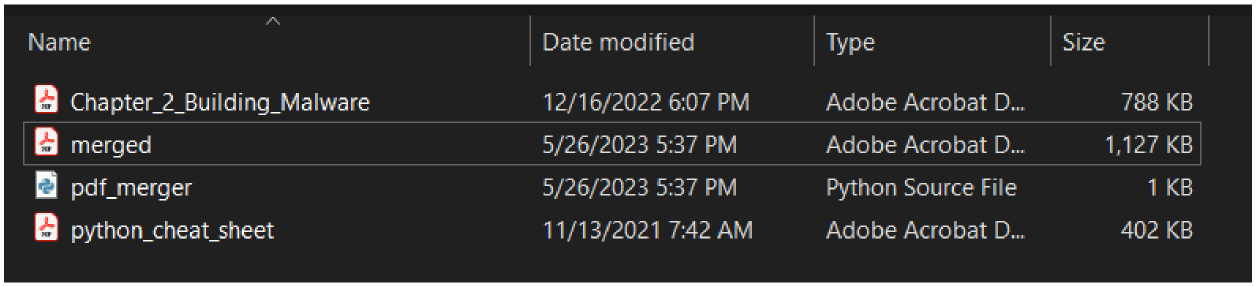
```
# save the output file
output_file.save('merged.pdf')
```

Running the Code

Let's run it now:

```
$ python pdf_merger.py python_cheat_sheet.pdf Chapter_2_Building_Malware.pdf
```

Keep in mind that the order is essential here, `python_cheat_sheet.pdf` will be the first in the newly generated PDF document. Here's the new PDF document:



Name	Date modified	Type	Size
Chapter_2_Building_Malware	12/16/2022 6:07 PM	Adobe Acrobat D...	788 KB
merged	5/26/2023 5:37 PM	Adobe Acrobat D...	1,127 KB
pdf_merger	5/26/2023 5:37 PM	Python Source File	1 KB
python_cheat_sheet	11/13/2021 7:42 AM	Adobe Acrobat D...	402 KB

Note that you can pass any number of PDF files, like this:

```
$ python pdf_merger.py file1.pdf file2.pdf file3.pdf file4.pdf
```

And the new merged PDF document will be created, named `merged.pdf`.

Excellent! With this, you now know how to merge multiple PDF files into a single document using PyMuPDF library in Python.

Chapter Wrap Up

As we reach the end of this introductory chapter, we have learned about the power and versatility of the PyMuPDF library in Python for handling PDF files. The capability to read, navigate, and extract text from PDFs, along with splitting and merging them, are foundational skills in PDF processing. We have familiarized ourselves with these functionalities, reinforcing the core understanding of PDF manipulation.

Looking forward, the knowledge and skills gained in this chapter will serve as the building blocks for more advanced PDF processing tasks. We'll delve deeper into extracting images, tables, and metadata, converting between different formats, and much more in the upcoming chapters. Rest assured, you are now well-equipped to face these advanced topics head-on. Stay tuned!

Note: *This is a free chapter released from the* ***Practical Python PDF Processing EBook****. If you wish to purchase the complete version (a total of 6 chapters), head to this link.*