PDF Manipulation with Python: A Comprehensive Guide to Building PDF Tools and Automation

medium.com/@meetjethwa3/pdf-manipulation-with-python-a-comprehensive-guide-to-building-pdf-tools-and-automation-c1b4564cdd8c

MEET JETHWA June 6, 2023

PDF Merger:

```
from PyPDF2 import PdfMerger

def merge_pdfs(input_files, output_file):
    merger = PdfMerger()



    for file in input_files:
        merger.append(file)



    merger.write(output_file)
    merger.close()

# Usage exampleinput_files = [, , ]output_file = (input_files, output_file)
```

PDF Splitter:

```
from PyPDF2 import PdfFileReader, PdfFileWriter

defsplit_pdf(input_file, output_directory, pages_per_split):
    input_pdf = PdfFileReader(input_file)
    total_pages = input_pdf.getNumPages()



for start inrange(0, total_pages, pages_per_split):
        end = start + pages_per_split
if end > total_pages:
            end = total_pages



        output_pdf = PdfFileWriter()
for page inrange(start, end):
            output_pdf.addPage(input_pdf.getPage(page))



        output_file = f"/split_-.pdf"
withopen(output_file, "wb") as f:
            output_pdf.write(f)

         input_file = output_directory = pages_per_split = split_pdf(input_file,
output_directory, pages_per_split)
```

PDF Text Extractor:

```
from PyPDF2 import PdfFileReader

def extract_text(input_file):
    input_pdf = PdfFileReader(input_file)
    total_pages = input_pdf.getNumPages()



    extracted_text = ""
    for page in range(total_pages):
        extracted_text += input_pdf.getPage(page).extractText()



    return extracted_text

# Usage exampleinput_file = text = (input_file)(text)
```

PDF to Image Converter:

```
from pdf2image import convert_from_path

defconvert_to_images(input_file, output_directory):
    images = convert_from_path(input_file)



for i, image inenumerate(images):
        output_file = f"/page_.png"
        image.save(output_file, "PNG")

 input_file = output_directory = convert_to_images(input_file, output_directory)
```

PDF Password Remover:

```
from PyPDF2 import PdfReader, PdfWriter

defremove_password(input_file, output_file, password):
withopen(input_file, "rb") as f:
        pdf = PdfReader(f)
if pdf.isEncrypted:
            pdf.decrypt(password)



withopen(output_file, "wb") as output:
            writer = PdfWriter()
for page in pdf.pages:
                writer.add_page(page)



            writer.write(output)

input_file = output_file = password = remove_password(input_file, output_file, password)
```

PDF Form Filler:

```
from pdfrw import PdfReader, PdfWriter

def fill_form(input_file, output_file, field_data):
    template = PdfReader(input_file)



    for page in template.pages:
        annotations = page.Annots or []
        for annotation in annotations:
            if annotation["/Subtype"] == "/Widget" and annotation["/FT"] == "/Tx":
                if annotation["/T"] in field_data:
                    annotation.update(
PdfReader(fill_pdf_data[field_data[annotation["/T"]]])
                    )



PdfWriter().write(output_file, template)

# Usage exampleinput_file = output_file = field_data = {    "Name": ,    : ,    : }
(input_file, output_file, field_data)
```

PDF Watermarker:

```
from PyPDF2 import PdfReader, PdfWriter
from reportlab.pdfgen import canvas

defadd_watermark(input_file, output_file, watermark_text, position=(), transparency=):
withopen(input_file, "rb") as file:
        pdf = PdfReader(file)
        writer = PdfWriter()



for page_num inrange(len(pdf.pages)):
            watermark = canvas.Canvas(f"watermark_page_.pdf")
            watermark.setFont("Helvetica", 40)
            watermark.setFillAlpha(transparency)
            watermark.rotate(45)
            watermark.drawString(position[0], position[1], watermark_text)
            watermark.save()



            page = pdf.pages[page_num]
            watermark_pdf = PdfReader(f"watermark_page_.pdf")
            watermark_page = watermark_pdf.pages[0]
            page.mergePage(watermark_page)



            writer.addPage(page)



withopen(output_file, "wb") as output:
            writer.write(output)

 input_file = output_file = watermark_text = add_watermark(input_file, output_file,
watermark_text)
```

PDF Metadata Editor:

```
from PyPDF2 import PdfFileReader, PdfFileWriter

defedit_metadata(input_file, output_file, title=, author=, keywords=):
withopen(input_file, "rb") as file:
        pdf = PdfFileReader(file)
        writer = PdfFileWriter()
        writer.cloneReaderDocumentRoot(pdf)



if title:
            writer.addMetadata({"/Title": title})
if author:
            writer.addMetadata({"/Author": author})
if keywords:
            writer.addMetadata({"/Keywords": keywords})



withopen(output_file, "wb") as output:
            writer.write(output)

 input_file = output_file = title = author = keywords = edit_metadata(input_file,
output_file, title, author, keywords)
```

PDF to Excel Converter:

```
import tabula

defconvert_to_excel(input_file, output_file):
    tabula.convert_into(input_file, output_file, output_format="xlsx")

 input_file = output_file = convert_to_excel(input_file, output_file)
```

PDF OCR (Optical Character Recognition):

```
import pytesseract
from PIL import Image
from pdf2image import convert_from_path

defperform_ocr(input_file):
    images = convert_from_path(input_file)
    text = ""



for image in images:
        text += pytesseract.image_to_string(image)



return text

input_file = text = perform_ocr(input_file)(text)
```

Python offers an extensive array of libraries and tools that empower developers and enthusiasts to manipulate PDF files with ease. In this blog, we have explored various Python projects for PDF manipulation, including merging, splitting, extracting text and images, modifying metadata, converting to Excel, and performing OCR. By harnessing the power of libraries such as PyPDF2, pdf2image, tabula-py, and pytesseract, we have seen how Python can revolutionize PDF workflows and automate mundane tasks.

With the knowledge gained from this guide, you now have the ability to create custom PDF solutions tailored to your specific needs. Whether you are working with large document collections, requiring watermarking for sensitive information, or extracting data for analysis, Python provides the flexibility and versatility to accomplish these tasks efficiently.

By combining your Python skills with the vast possibilities of PDF manipulation, you can enhance productivity, streamline workflows, and create innovative solutions. Whether you are a developer, data analyst, or an enthusiast exploring the realm of PDF management, Python empowers you to unlock the true potential of PDF files.

We hope this blog has provided you with valuable insights and practical code examples to embark on your own PDF manipulation journey with Python. Remember to explore the official documentation of the libraries mentioned, experiment with different techniques, and adapt them to suit your specific requirements.

Now, armed with this newfound knowledge, it’s time to dive into the world of Python-powered PDF manipulation and unlock endless possibilities for your projects. Happy coding and may your PDF endeavors be successful!

Python from scratch and hands on with all basics which even a layman can understand and learn python. Python can be…
