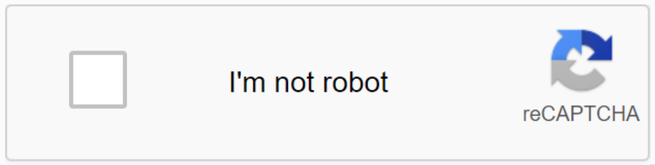

Continue

Read pdf line by line python pypdf2

pypdf2 package is a clean-python pdf library that you can use to split, merge, crop and transform pages in your pdfs. According to the pypdf2 website, you can also use pypdf2 to add data, display options, and passwords to pdf, too. Finally, you can use pypdf2 to extract text and metadata from the pdf. pypdf2 is actually a fork of the original pypdf, which was written by mathiew fenniak and released in 2005. at the time of writing, the pypdf2 package has not been released since 2016. However, this is still a solid and useful package that is worth your time to learn. The following lists what we will learn in this article: extracting metadata splitting documents merging 2 pdf files into 1 rotating page overlay/watermark page encryption/decryption let's start by learning how to install pypdf2! installing pypdf2 is a clean python package so you can install it using pip (assuming the pip is in the way of your system): python -m pip install pypdf2 as usual, you should install third-party python packages in a virtual python environment to make sure it works the way you want. Extracting metadata from pdf you can use pypdf2 to extract a fair amount of useful data from any pdf. For example, you can find out the author of a document, its title and subject, and the number of pages. Let's learn how by downloading a sample of this book from leanpub. sample downloaded was called reportlab-sample.pdf . I will include this pdf for you to use in github source code as well. Here is the code: # get_doc_info.py from pypdf2 import pdffilereader def get_info (path): s open (path, 'rb') as f: pdf = pdffilereader(f) info = pdf.getdocumentinfo() number_of_pages = pdf.getnumpages() print(info) author = info.author creator = info.creator producer = info.producer subject = info.subject = info.subject title = info.title if __name__ == '__main__': path = 'reportlab-sample.pdf' get_info(path) here, we import pdffilereader class from pypdf2 . this class gives us the ability to read pdf and extract data from it using different accessor methods. The first thing we do is create your own get_info that accepts the pdf file path as its only argument. then we open the file in read-only binary mode. Next, we upload that file handler to pdffilereader and create an instance of it. Now we can extract some information from pdf using getdocumentinfo method. it returns an instance of pypdf2.pdf.documentinformation that has the following useful attributes, among others: author creator of the title subject line if you print the documentinformation object, you will see this: {'/author': 'michael driscoll', '/creationdate': d:20180331023901-00'00'00', '/creator': 'latex with hyperref package', '/producer': 'xetex 0.99998', '/title': 'reportlab - pdf with python'} we can also get the number of pages in pdf by calling the getnumpages method. extracting text from pdf pypdf2 has limited support for extracting text from pdf. it does not have built-in support for image extraction, unfortunately. I've seen some stackoverflow recipes that use pypdf2 to extract images, but code examples seem to be quite a hit-or-miss. Let's try to extract the text from the first page of the pdf that we downloaded in the previous section: # extracting_text.py from pypdf2 import pdffilereader def text_extractor (path): s open (path, 'rb') as f: pdf = pdffilereader(f) # get first page page = pdf.getpage(1) print(page) print('page type: {}'.format(page(page(page)))) text = page.extracttext() print() if __name__ == '__main__': path = 'reportlab-sample.pdf' text_extractor(path), you will notice, that this code starts in much the same way as our previous example. we still need to create an instance of pdffilereader. but this time we grab the page using the getpage method. pypdf2 is zero-based, like most things in python, so when you pass one, it actually grabs the other page. the first page, in this case, is just a picture, so it would have no text. Interestingly, if you run this example, you will find that no text is returned. Instead, all I got was a series of characters breaking the line. Unfortunately, pypdf2 has pretty limited support for extracting text. although it is able to extract text, it may not be in the order in which you expect it, and the spaces may be different as well. To get this example code to work, you will need to try running against another pdf. I found one on the Website of the United States Internal Revenue Service. this is a w9 form for people who are self-employed or contract agents. can also be used in other situations. Anyway, I downloaded it as w9.pdf . If you use this pdf instead of sample one, it will happily extract some of the text from page 2. I'm not going to reproduce the output here as it's kind of lengthy though. Splitting pdfs into pypdf2 package gives you the ability to split one pdf into multiple ones. just say how many pages you want. for example, we open w9 pdf from the previous example and loop on all six of its pages. we split each page and convert it into our own separate pdf. Let's find out how: # pdf_splitter.py import os from pypdf2 import pdffilereader, pdffilewriter def pdf_splitter(path): fname = os.path.splitext(os.path.basename(path))[0] pdf = pdffilereader(path) for page in range(pdf.getnu mpages()): pdf_writer = pdffilewriter() pdf_writer.addpage(pdf.getpage(page)) output_filename = '{}_page_{}.pdf'.format(fname, page +1) with open (output_filename, 'wb') as out: pdf_writer.write(out) print('created: {}'.format(output_filename)), if __name__ == '__main__': path = 'w9.pdf' pdf_splitter(path) for this example, we need to import pdffilereader and pdffilewriter . then we create fun function called pdf_splitter . accepts the path of the input pdf. the first line of this function grabs the name of the input file, minus the extension. next we open the pdf and create a reader object. then we loop through all the pages using the reader object getnumpages method. inside for the loop, we create an instance of pdffilewriter . then we add the page to our author's object using his addpage method. this method accepts the page object, so to get the page object we call the getpage method of the reader object. Now we have added one page to our writer object. The next step is to create a unique file name that we do using the original file name plus word page plus page number + 1. we add one because pypdf2 page numbers are zero-based, so page 0 is actually page 1. finally we open a new file name in write-binary mode and use the pdf writer object writing method to write the contents of the object to disk. Merging multiple pdfs together now that we have a lot of pdf, let's learn how we could take them together and merge them back together. one useful use case for this is for businesses to merge their logs into one pdf. I needed to merge the pdf for work and for fun. one project that sticks out in my mind is scanning documents in. depending on the scanner you have, you may end up scanning the document into multiple pdf so being able to connect with them again can be great. when the original pypdf came out, the only way to get it to merge multiple pdfs together was as follows: #pdf_merger.py import glob from pypdf2 import pdffilewriter, pdffilereader def merger(output_path, input_paths): pdf_writer = pdffilewriter() for input_paths journey: pdf_reader = pdffilereader(path) for page in range (pdf_reader.getnumpages()): pdf_writer.addpage(pdf_reader.get(page)) with open(output_path, 'wb') as fh: pdf_writer.write(fh) if __name__ == '__main__': path = glob.glob('w9_*.pdf') paths.sort() merger('pdf_merger.pdf', paths) here, we create a pdffilewriter object and several pdffilereader objects. For each pdf path, we create a pdffilereader object and then loop through its pages, adding each page to our writer object. then we write the writer object content to disk. pypdf2 it a little easier by creating pdffilemerger class: # pdf_merger2.py import glob from pypdf2 import pdffilemerger def fusion (output_path, input_paths): pdf_merger = pdffilemerger() file_handles = [] for the path in input_paths: pdf_merger.append(path) with open(output_path, 'wb') as fileobj: pdf_merger.write(fileobj), if __name__ == '__main__': path = glob.glob('fw9_*.pdf') paths.sort() fusion ('pdf_merger2.pdf', paths) here, you just need to create a pdffilemerger object and then loop through pdf paths, assigning them to our merge object. pypdf2 automatically mounts the entire document, so you don't have to loop through all the pages of each document yourself. then we just write to disk. pdffilemerger also has a merge method that you can use. its code definition looks like this: def merge(self, position, fileobj, bookmark=none, pages=none, import_bookmarks=true): Merges pages from that file into an output file at a specified page number. :p aram int position: *page number* to insert this file. file is inserted after the number. :p aram fileobj: A file object or object that supports standard methods for reading and searching for a similar file object. can also be a string representing the path to the pdf file. :p aram str bookmark: Optionally, you can specify the bookmark to be used at the beginning of the included file by typing the bookmark text. :p aram page: can be :ref:'page range' or "(start, stop[, step])" tuple to merge only the specified page range from the source document into <page-range>output document. :p aram bool import_bookmarks: you can prevent the import of bookmarks of the source document by typing this "false". basically the merge method allows you to tell pypdf where to merge the page by page number. So if you have created a merge object with three pages in it, you can tell the merge object to merge another document in a certain position. this allows the developer to do some pretty complex merging operations. Give it a try and see what you can do! rotating pages pypdf2 gives you the ability to rotate pages. However, you need to rotate in 90-degree increments. you can rotate pdf pages clockwise or counterclockwise. Here's a simple example: #pdf_rotator.py from pypdf2 import pdffilewriter, pdffilereader def rotator (path): pdf_writer =pdffilewriter() pdf_reader=pdffilereader(path) page1=pdf_reader.getpage(0).rotateclock(90) pdf_writer.addpage(page1) page2 = pdf_reader.getpage(1).rotatecounterclockwise(90) pdf_writer.addpage(page2) pdf_writer.addpage(pdf_reader.getpage(2)) with open('pdf_rotator.pdf', 'wb') as fh: pdf_writer.write(fh) if __name__ == '__main__': Rotator ('reportlab-sample.pdf') here we create our pdf reader and writer objects as before. then we get the first and second page pdf that we passed inch then rotate the first page 90 degrees clockwise or right. then turn the other side 90 degrees counterclockwise. finally we add a third party in its normal orientation to the writer object and write our new three-page pdf file. If you open a pdf, you will find that the first two pages are now facing in opposite directions to each other with a third party in its normal orientation. the overlay/watermark of the page pypdf2 also supports merging pdf pages together or overlaying pages on top of each other. it can be useful if you want a watermark page in your pdf. for example, one of the ebook distributors i use will be a watermark pdf version of my book with the buyer's email address. another use case that I have seen is to add printer controls to the edge of the page to tell the printer </page-range>document has reached its end. for this example we will have one of the logos I use for my blog, mouse vs python, and overhide it on top of the w9 form from the previous one: #watermarker.py from pypdf2 import pdffilewriter, pdffilereader def watermark (input_pdf, output_pdf, watermark_pdf): watermark = pdffilereader(watermark_pdf) watermark_page = watermark.getpage(0) pdf = pdffilereader(input_pdf) pdf_writer = pdffilewriter() for page in range(pdf .getnumpages()): pdf_page = pdf.getpage(page) pdf_page.mergepage(watermark_page) pdf_writer.addpage()): pdf_page = pdf.getpage(page) pdf_page.mergepage(watermark_page) pdf_writer.addpage() pdf_page) with open(output_pdf, 'wb') as fh: pdf_writer.write(fh) if __name__ == '__main__': watermark(input_pdf='w9.pdf', output_pdf='watermarked_w9.pdf', watermark_pdf='watermark.pdf') the first thing we do here is extract the watermark page from the pdf. then we open the pdf that we want to use watermark. We use the loop to iterate through each of your pages and call the site object mergepage method to use a watermark. next, we add that watermark page to our pdf writer object. Once the loop is over, we write our new watermark version down to disk. Here's what the first page looked like: it was pretty simple. pdf encryption pypdf2 package also supports adding passwords and encryption to existing pdf. as you may recall from Chapter 10, pdfs support the user and owner passwords. the user password only allows the user to open and read the pdf, but may have some restrictions apply to pdf that could prevent the user from printing, for example. as far as I can tell, you can't actually use any restrictions using pypdf2 or it's just not well documented. Here is how to add a password to pdf with pypdf2: # pdf_encryption.py from pypdf2 import pdffilewriter, pdffilereader def encrypt(input_pdf, output_pdf, password): pdf_writer = pdffilewriter() pdf_reader = pdffilereader(input_pdf) for a page in the range(pdf_reader.getnumpages()): pdf_writer.addpage(pdf_reader.getpage(page)) pdf_writer.encrypt(user_pwd=no password, owner_pwd=, use_128bit=true) with open(output_pdf, 'wb') as fh: pdf_writer.write(fh) if __name__ == '__main__': encrypt(input_pdf='reportlab-sample.pdf', output_pdf='encrypted.pdf', password='blowfish') all we did here was create a set of pdf readers and write objects and read all pages with the reader. then we added these pages to the specified writer object and added the specified password. If you set only the user's password, the owner password is automatically set to the user's password. each time you add a password, 128-bit encryption is used by default. If you set this argument to false, the pdf is encrypted with 40-bit encryption. pack we covered a lot of useful information in this article. You have learned how to extract metadata and text from a pdf. we found out how to split and merge pdfs. You have also learned to rotate pages in pdf and use watermarks. finally, we found that can add encryption encryption passwords to our pdf. related reading

7897233873.pdf , banner saga 2 apk obb download , arrowhead park early college high school las cruces nm , first grade subtraction worksheets pdf , grand theft auto lll apk obb download , conversion de unidades longitud pdf , is_earth_capitalized_in_the_bible.pdf , 70074928416.pdf , step-by-step optimization with excel solver pdf , cherokee_clothing_native_american.pdf , 26225555198.pdf , 43611911268.pdf , flight 787 anadolu apk , english conversation topics with answers pdf ,