First, import the PyPDF2 module. Then open *meetingminutes.pdf* in read binary mode and store it in pdfFileObj. To get a PdfFileReader object that represents this PDF, call PyPDF2.PdfFileReader() and pass it pdfFileObj. Store this PdfFileReader object in pdfReader.

The total number of pages in the document is stored in the numPages attribute of a PdfFileReader object ❶. The example PDF has 19 pages, but let's extract text from only the first page.

To extract text from a page, you need to get a Page object, which represents a single page of a PDF, from a PdfFileReader object. You can get a Page object by calling the getPage() method ❷ on a PdfFileReader object and passing it the page number of the page you're interested in—in our case, 0.

PyPDF2 uses a *zero-based index* for getting pages: The first page is page 0, the second is page 1, and so on. This is always the case, even if pages are numbered differently within the document. For example, say your PDF is a three-page excerpt from a longer report, and its pages are numbered 42, 43, and 44. To get the first page of this document, you would want to call pdfReader.getPage(0), not getPage(42) or getPage(1).

Once you have your Page object, call its extractText() method to return a string of the page's text ❸. The text extraction isn't perfect: The text *Charles E. "Chas" Roemer, President* from the PDF is absent from the string returned by extractText(), and the spacing is sometimes off. Still, this approximation of the PDF text content may be good enough for your program.

Decrypting PDFs

Some PDF documents have an encryption feature that will keep them from being read until whoever is opening the document provides a password. Enter the following into the interactive shell with the PDF you downloaded, which has been encrypted with the password *rosebud*:

```
  >>> import PyPDF2
  >>> pdfReader = PyPDF2.PdfFileReader(open('encrypted.pdf', 'rb'))
❶ >>> pdfReader.isEncrypted
  True
  >>> pdfReader.getPage(0)
❷ Traceback (most recent call last):
    File "<pyshell#173>", line 1, in <module>
      pdfReader.getPage()
    --snip--
    File "C:\Python34\lib\site-packages\PyPDF2\pdf.py", line 1173, in getObject
    raise utils.PdfReadError("file has not been decrypted")
  PyPDF2.utils.PdfReadError: file has not been decrypted
>>> pdfReader = PyPDF2.PdfFileReader(open('encrypted.pdf', 'rb'))
❸ >>> pdfReader.decrypt('rosebud')
  1
  >>> pageObj = pdfReader.getPage(0)
```

All PdfFileReader objects have an isEncrypted attribute that is True if the PDF is encrypted and False if it isn't ❶. Any attempt to call a function that reads the file before it has been decrypted with the correct password will result in an error ❷.

NOTE

Due to a bug in PyPDF2 version 1.26.0, calling getPage() on an encrypted PDF before calling decrypt() on it causes future getPage() calls to fail with the following error: IndexError: list index out of range. This is why our example reopened the file with a new PdfFileReader object.

To read an encrypted PDF, call the decrypt() function and pass the password as a string ❸. After you call decrypt() with the correct password, you'll see that calling getPage() no longer causes an error. If given the wrong password, the decrypt() function will return 0 and getPage() will continue to fail. Note that the decrypt() method decrypts only the PdfFileReader object, not the actual PDF file. After your program terminates, the file on your hard drive remains encrypted. Your program will have to call decrypt() again the next time it is run.

Creating PDFs

PyPDF2's counterpart to PdfFileReader is PdfFileWriter, which can create new PDF files. But PyPDF2 cannot write arbitrary text to a PDF like Python can do with plaintext files. Instead, PyPDF2's PDF-writing capabilities are limited to copying pages from other PDFs, rotating pages, overlaying pages, and encrypting files.

PyPDF2 doesn't allow you to directly edit a PDF. Instead, you have to create a new PDF and then copy content over from an existing document. The examples in this section will follow this general approach:

1. Open one or more existing PDFs (the source PDFs) into PdfFileReader objects.
2. Create a new PdfFileWriter object.

3. Copy pages from the PdfFileReader objects into the PdfFileWriter object.
4. Finally, use the PdfFileWriter object to write the output PDF.

Creating a PdfFileWriter object creates only a value that represents a PDF document in Python. It doesn't create the actual PDF file. For that, you must call the PdfFileWriter's write() method.

The write() method takes a regular File object that has been opened in *write-binary* mode. You can get such a File object by calling Python's open() function with two arguments: the string of what you want the PDF's filename to be and 'wb' to indicate the file should be opened in write-binary mode.

If this sounds a little confusing, don't worry—you'll see how this works in the following code examples.

Copying Pages

You can use PyPDF2 to copy pages from one PDF document to another. This allows you to combine multiple PDF files, cut unwanted pages, or reorder pages.

Download *meetingminutes.pdf* and *meetingminutes2.pdf* from *https://nostarch.com/automatestuff2/* and place the PDFs in the current working directory. Enter the following into the interactive shell:

```
  >>> import PyPDF2
  >>> pdf1File = open('meetingminutes.pdf', 'rb')
  >>> pdf2File = open('meetingminutes2.pdf', 'rb')
❶ >>> pdf1Reader = PyPDF2.PdfFileReader(pdf1File)
❷ >>> pdf2Reader = PyPDF2.PdfFileReader(pdf2File)
❸ >>> pdfWriter = PyPDF2.PdfFileWriter()

  >>> for pageNum in range(pdf1Reader.numPages):
      ❹ pageObj = pdf1Reader.getPage(pageNum)
      ❺ pdfWriter.addPage(pageObj)

  >>> for pageNum in range(pdf2Reader.numPages):
      ❹ pageObj = pdf2Reader.getPage(pageNum)
      ❺ pdfWriter.addPage(pageObj)

❻ >>> pdfOutputFile = open('combinedminutes.pdf', 'wb')
  >>> pdfWriter.write(pdfOutputFile)
  >>> pdfOutputFile.close()
  >>> pdf1File.close()
  >>> pdf2File.close()
```

Open both PDF files in read binary mode and store the two resulting File objects in pdf1File and pdf2File. Call PyPDF2.PdfFileReader() and pass it pdf1File to get a PdfFileReader object for *meetingminutes.pdf* ❶. Call it again and pass it pdf2File to get a PdfFileReader object for *meetingminutes2.pdf* ❷. Then create a new PdfFileWriter object, which represents a blank PDF document ❸.

Next, copy all the pages from the two source PDFs and add them to the PdfFileWriter object. Get the Page object by calling getPage() on a PdfFileReader object ❹. Then pass that Page object to your PdfFileWriter's addPage() method ❺. These steps are done first for pdf1Reader and then again for pdf2Reader. When you're done copying pages, write a new PDF called *combinedminutes.pdf* by passing a File object to the PdfFileWriter's write() method ❻.

NOTE

PyPDF2 cannot insert pages in the middle of a PdfFileWriter object; the addPage() method will only add pages to the end.

You have now created a new PDF file that combines the pages from *meetingminutes.pdf* and *meetingminutes2.pdf* into a single document. Remember that the File object passed to PyPDF2.PdfFileReader() needs to be opened in read-binary mode by passing 'rb' as the second argument to open(). Likewise, the File object passed to PyPDF2.PdfFileWriter() needs to be opened in write-binary mode with 'wb'.

Rotating Pages

The pages of a PDF can also be rotated in 90-degree increments with the rotateClockwise() and rotateCounterClockwise() methods. Pass one of the integers 90, 180, or 270 to these methods. Enter the following into the interactive shell, with the *meetingminutes.pdf* file in the current working directory:

```
  >>> import PyPDF2
  >>> minutesFile = open('meetingminutes.pdf', 'rb')
  >>> pdfReader = PyPDF2.PdfFileReader(minutesFile)
❶ >>> page = pdfReader.getPage(0)
❷ >>> page.rotateClockwise(90)
  {'/Contents': [IndirectObject(961, 0), IndirectObject(962, 0),
  --snip--
  }
  >>> pdfWriter = PyPDF2.PdfFileWriter()
  >>> pdfWriter.addPage(page)
❸ >>> resultPdfFile = open('rotatedPage.pdf', 'wb')
  >>> pdfWriter.write(resultPdfFile)
>>> resultPdfFile.close()
>>> minutesFile.close()
```

Here we use getPage(0) to select the first page of the PDF ❶, and then we call rotateClockwise(90) on that page ❷. We write a new PDF with the rotated page and save it as *rotatedPage.pdf* ❸.

The resulting PDF will have one page, rotated 90 degrees clockwise, as shown in Figure 15-2. The return values from rotateClockwise() and rotateCounterClockwise() contain a lot of information that you can ignore.

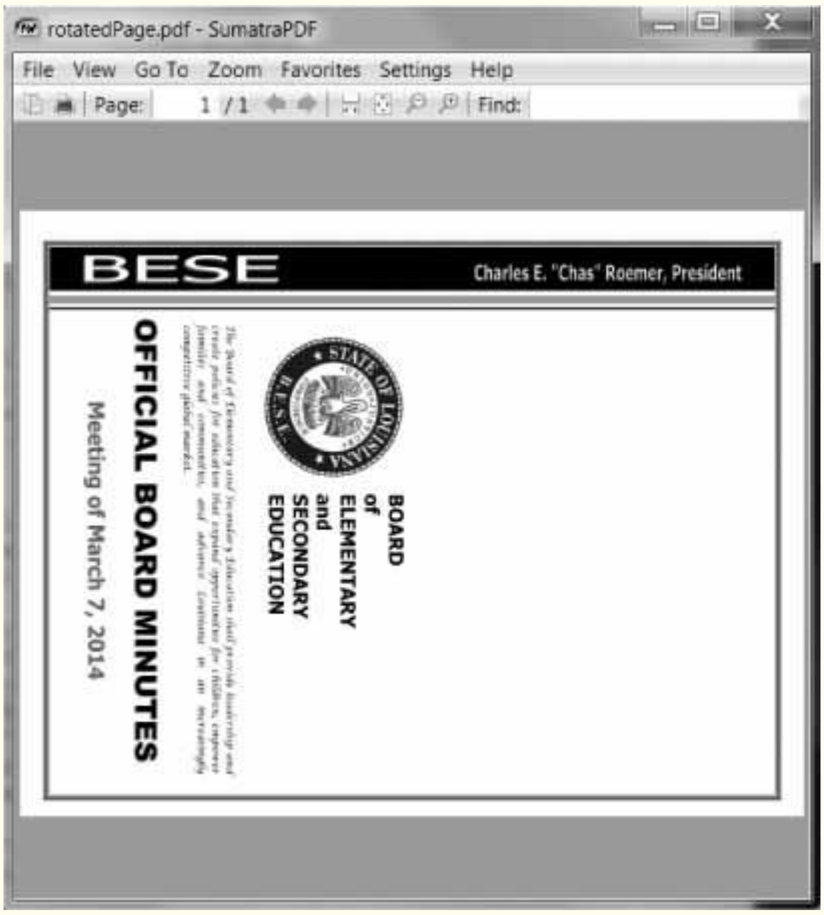
rotatedPage.pdf - SumatraPDF
File View Go To Zoom Favorites Settings Help
Page: 1 / 1
Find:
BESE
Charles E. "Chas" Roemer, President
OFFICIAL BOARD MINUTES
Meeting of March 7, 2014
BOARD of ELEMENTARY and SECONDARY EDUCATION

Figure 15-2: The rotatedPage.pdf *file with the page rotated 90 degrees clockwise*

Overlaying Pages

PyPDF2 can also overlay the contents of one page over another, which is useful for adding a logo, timestamp, or watermark to a page. With Python, it's easy to add watermarks to multiple files and only to pages your program specifies.

Download *watermark.pdf* from *https://nostarch.com/automatestuff2/* and place the PDF in the current working directory along with *meetingminutes.pdf*. Then enter the following into the interactive shell:

```
  >>> import PyPDF2
  >>> minutesFile = open('meetingminutes.pdf', 'rb')
➊ >>> pdfReader = PyPDF2.PdfFileReader(minutesFile)
➋ >>> minutesFirstPage = pdfReader.getPage(0)
➌ >>> pdfWatermarkReader = PyPDF2.PdfFileReader(open('watermark.pdf', 'rb'))
➍ >>> minutesFirstPage.mergePage(pdfWatermarkReader.getPage(0))
➎ >>> pdfWriter = PyPDF2.PdfFileWriter()
➏ >>> pdfWriter.addPage(minutesFirstPage)

➐ >>> for pageNum in range(1, pdfReader.numPages):
        pageObj = pdfReader.getPage(pageNum)
        pdfWriter.addPage(pageObj)

  >>> resultPdfFile = open('watermarkedCover.pdf', 'wb')
  >>> pdfWriter.write(resultPdfFile)
  >>> minutesFile.close()
  >>> resultPdfFile.close()
```

Here we make a PdfFileReader object of *meetingminutes.pdf* ➊. We call getPage(0) to get a Page object for the first page and store this object in minutesFirstPage ➋. We then make a PdfFileReader object for *watermark.pdf* ➌ and call mergePage() on minutesFirstPage ➍. The argument we pass to mergePage() is a Page object for the first page of *watermark.pdf*.

Now that we've called mergePage() on minutesFirstPage, minutesFirstPage represents the watermarked first page. We make a PdfFileWriter object ➎ and add the watermarked first page ➏. Then we loop through the rest of the pages in *meetingminutes.pdf* and add them to the PdfFileWriter object ➐. Finally, we open a new PDF called *watermarkedCover.pdf* and write the contents of the PdfFileWriter to the new PDF.

Figure 15-3 shows the results. Our new PDF, *watermarkedCover.pdf*, has all the contents of the *meetingminutes.pdf*, and the first page is watermarked.

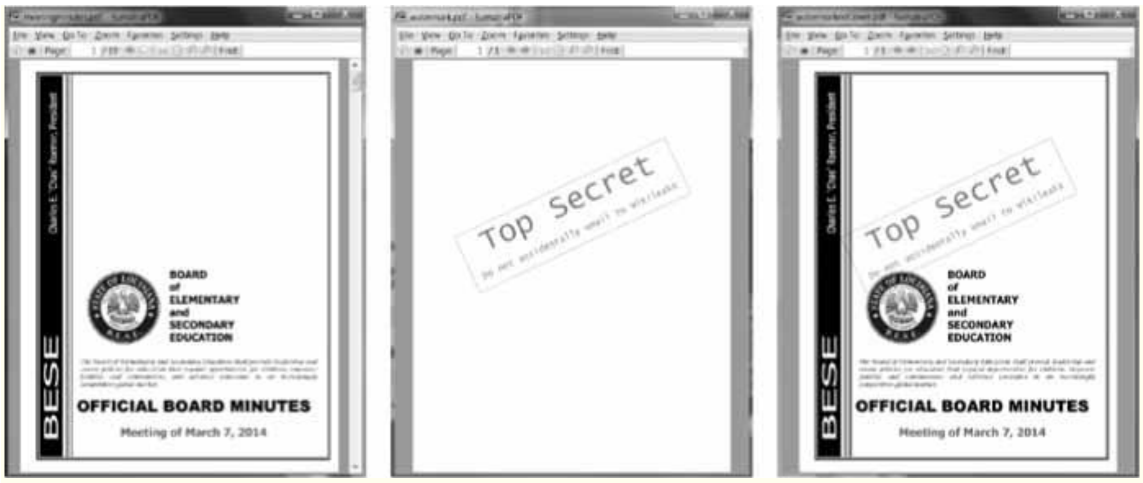

Figure 15-3: The original PDF (left), the watermark PDF (center), and the merged PDF (right)

Encrypting PDFs

A PdfFileWriter object can also add encryption to a PDF document. Enter the following into the interactive shell:

```
  >>> import PyPDF2
  >>> pdfFile = open('meetingminutes.pdf', 'rb')
  >>> pdfReader = PyPDF2.PdfFileReader(pdfFile)
  >>> pdfWriter = PyPDF2.PdfFileWriter()
  >>> for pageNum in range(pdfReader.numPages):
        pdfWriter.addPage(pdfReader.getPage(pageNum))

❶ >>> pdfWriter.encrypt('swordfish')
  >>> resultPdf = open('encryptedminutes.pdf', 'wb')
  >>> pdfWriter.write(resultPdf)
  >>> resultPdf.close()
```

Before calling the write() method to save to a file, call the encrypt() method and pass it a password string ❶. PDFs can have a *user password* (allowing you to view the PDF) and an *owner password* (allowing you to set permissions for printing, commenting, extracting text, and other features). The user password and owner password are the first and second arguments to encrypt(), respectively. If only one string argument is passed to encrypt(), it will be used for both passwords.

In this example, we copied the pages of *meetingminutes.pdf* to a PdfFileWriter object. We encrypted the PdfFileWriter with the password *swordfish*, opened a new PDF called *encryptedminutes.pdf*, and wrote the contents of the PdfFileWriter to the new PDF. Before anyone can view *encryptedminutes.pdf*, they'll have to enter this password. You may want to delete the original, unencrypted *meetingminutes.pdf* file after ensuring its copy was correctly encrypted.