PDF e Word em Python

Roger Tiago

Abril 2016

Sumário

0.1 Introdução

Este documento visa auxiliar o trabalho com arquivos PDF e .docx utilizando Python.

0.2 Documentos

Documentos em PDF e .docx são arquivos binários, oque dificulta o trabalho com esses arquivos, porem o python conta com módulos para facilitar estes processos.

Capítulo 1

PDF

1.1 Documentos em PDF

Documentos em PDF são arquivos .pdf, PDF's podem conter textos, imagens e vídeos, aqui será focado em editar e ler textos de aquivos PDF, para isso usaremos o módulo PyPDF2.

1.1.1 Extraindo textos de PDF's

O módulo PyPDF2 não consegue extrair imagens, gráficos e outras mídias dos arquivos, porém pode extrair textos e retornar como uma string.

Para extrair um texto de um PDF é necessário a importação do módulo PyPDF2, cria-se um objeto para ler o pdf em modo binário e armazenar, é possivel obter o número de páginas do documento que começam a ser contadas a partir do 0, para extrair o texto propriamente dito usamo a função .getPage(n) no objeto criado anteriormente onde n representa a página que desejamos o texto(contada a partir do 0), após isso usa-se a função .extractText() no objeto que contém o pdf já converito em binário e o indice da pagina, será mostrada uma string contendo o texto.

Código exemplo

```
>>> import PyPDF2
>>> pdfFileObj = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
>>> pdfReader.numPages
>>> pageObj = pdfReader.getPage(0)
>>> pageObj.extractText()
```

1.1.2 Descriptografando PDF's

Alguns PDF's contam com um sistema de criptografia, e nao será possível ler o documento até que seja descriptografado. Todo objeto PdfFileReader conta com um atributo isEncrypted que se tiver o valor True nao será possível ler o PDF. Neste caso para ler um PDF criptogradfado basta usar a função .decrypt() no objeto com a senha do documento como argumento em string, se a senha estiver correta sera possivel acessar o documento, caso contrario ainda haverá erros.A função .decrypt() funciona apenas para o objeto e nao para o arquivo em si.

Código exemplo

```
>>> import PyPDF2
>>> pdfReader = PyPDF2.PdfFileReader(open('encrypted.pdf','rb'))
>>> pdfReader.isEncrypted
>>> pdfReader.getPage(0)
```

1.1.3 Criando PDF's

PyPDF2 pode criar novos PDF's, no entanto não é possível escrever arbitrariamente, apenas copiar páginas de outros PDF's, rotacionar, sobrepor e criptografar. PyPDF2 não pode editar diretamente um PDF, é necessário criar um novo PDF e importar textos já existentes. Para fazer isso é necessário abrir um ou mais PDF's fontes como objetos com PdfFileReader, criar um novo objeto PdfFileWriter, copiar as páginas do objeto PdfFileReader para o objeto PdfFileWriter, finalmente usamo o objeto PdfFileWriter para escrever o PDF final. O objeto PdfFileWriter não criar o arquivo em si, para isto é necessário usar o método write().

1.1.4 Copiando páginas

É possível utilizar o PyPDF2 para copiar páginas de um documento PDF para outro, isto permite combinar múltiplos arquivos PDF's, cortar páginas indesejadas e reordenas as páginas

Código exemplo

```
>>> import PyPDF2
>>> pdf1File = open('meetingminutes.pdf', 'rb')
>>> pdf2File = open('meetingminutes2.pdf', 'rb')
```

```
>>> pdf1Reader = PyPDF2.PdfFileReader(pdf1File)
>>> pdf2Reader = PyPDF2.PdfFileReader(pdf2File)
>>> pdfWriter = PyPDF2.PdfFileWriter()
>>> for pageNum in range(pdf1Reader.numPages):
        pageObj = pdf1Reader.getPage(pageNum)
        pdfWriter.addPage(pageObj)
>>> for pageNum in range(pdf2Reader.numPages):
        pageObj = pdf2Reader.getPage(pageNum)
        pdfWriter.addPage(pageObj)
>>> pdfOutputFile = open('combinedminutes.pdf', 'wb')
>>> pdfWriter.write(pdfOutputFile)
>>> pdfOutputFile.close()
>>> pdf1File.close()
>>> pdf2File.close()
```

1.1.5 Rotacionando páginas

É possível rotacionar páginas de um PDF em incrementos de 90º com os métodos rotateClockWise() e rotateCounterClockWise() passando o argumento 90, 180 ou 270.

Código exemplo

```
>>> import PyPDF2
>>> minutesFile = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(minutesFile)
>>> page = pdfReader.getPage(0)
>>> page.rotateClockwise(90)
>>> pdfWriter = PyPDF2.PdfFileWriter()
>>> pdfWriter.addPage(page)
>>> resultPdfFile = open('rotatedPage.pdf', 'wb')
>>> pdfWriter.write(resultPdfFile)
>>> resultPdfFile.close()
>>> minutesFile.close()
```

1.1.6 Sobrepondo páginas

Utilizando PyPDF2 é possível sobrepor páginas, facilmente adioconando logos ou marcas d'água a multiplas páginas.

Código exemplo

```
>>> import PyPDF2
>>> minutesFile = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(minutesFile)
>>> minutesFirstPage = pdfReader.getPage(0)
>>> pdfWatermarkReaderPyPDF2.PdfFileReader(open('watermark.pdf', 'rb'))
>>> minutesFirstPage.mergePage(pdfWatermarkReader.getPage(0))
>>> pdfWriter = PyPDF2.PdfFileWriter()
>>> pdfWriter.addPage(minutesFirstPage)

>>> 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()
```

1.1.7 Criptografando

O objeto PdfFileWriter pode adicionar uma senha a um arquivo PDF com o método .encrypt() passando como argumento a senha desejada.

Código exemplo

```
>>> 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))

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

Capítulo 2

Word

2.1 Documentos em .docx

Python pode criar e modificar documentos .docx, para isso é usado o módulo python-docx. Documentos .docx possuem muita estrutura, as quais são representadas por 3 diferentes tipos de data no módulo Python-Docx. No nível mais alto o objeto Document contém o documento inteiro e uma lista do objeto Paragraph para os parágrafos no documento(um paragrafo novo é criado a cada 'enter' teclado), cada um desses objetos Paragraph contém um ou mais objetos Run. Um texto em .docx é mais que uma string, ele contém fonte, cor e tamanho, um objeto Run é necessário sempre que o texto muda de estilo;

2.1.1 Lendo Documentos em .docx

Para lermos um documento word usamo o módulo docx, chamando a função docx.Document() e passando o nome do arquivo como argumento obtemos o objeto Document, que contém o atributo paragraph que por sua vez é uma lista de objetos Paragraph. Usando a função len(doc.paragraphs) obtemos o número de parágrafos no documento. Cada objeto Paragraph contém uma string que representa o texto neste parágrafo. Cara objeto Paragraph contém um atributo Run que é uma lista de objetos Run estes objetos contém um atributo com o texto nessa Run em particular. Uma Run muda sempre que apertamos 'Enter' ou 'Tab'.

Código Exemplo

```
>>> import docx
>>> doc = docx.Document
```

```
>>> len(doc.paragraphs)
>>> doc.paragraphs[0].text
>>> doc.paragraphs[1].text
>>> len(doc.paragraphs[1].runs)
>>> doc.paragraphs[1].runs[0].text
>>> doc.paragraphs[1].runs[1].text
>>> doc.paragraphs[1].runs[2].text
>>> doc.paragraphs[1].runs[3].text
```

2.1.2 Textos inteiros de documentos .docx

Se precisarmos apenas do texto em um documento .docx sem nos importarmos com as configurações de estilização do texto, podemos usar a função getText() passando o nome do arquivo como argumento. A função abre o documento, passa por todos os objetos Paragraph na lista paragraphs e acrescenta o texto na lista fullText.

Código Exemplo

```
import docx
def getText(filename):
    doc = docx.Document(filename)
    fullText = []
    for para in doc.paragraphs:
        fullText.append(para.text)
    return '\n'.join(fullText)
```

2.1.3 Escrevendo documentos em .docx

Usando docx.Document criamos um objeto de documento word vazio, com o método add_paragraph() é possível adicionar um parágrafo ao documento e retorna uma referência ao parágrafo que foi adicionado, ao terminar para salvar o arquivo usamos o método save() passando como argumento o nome do arquivo desejado. É possível adicionar outros parágrafos com o método add_paragraph() ou ainda adicionar textos ao final de um parágrafo existente com o método add_run(), ambos os métodos aceitam um segundo argumento de estilização de texto.

Código Exemplo

```
>>> import docx
>>> doc = docx.Document()
>>> doc.add_paragraph('Hello world!')
<docx.text.Paragraph object at 0x0000000003B56F60>
>>> paraObj1 = doc.add\_paragraph('This is a second paragraph.')
>>> paraObj2 = doc.add\_paragraph('This is a yet another paragraph.')
>>> paraObj1.add_run(' This text is being added to
the second paragraph.')
<docx.text.Run object at 0x0000000003A2C860>
>>> doc.add_paragraph('Hello world!', 'Title')
>>> doc.save('multipleParagraphs.docx')
```

2.1.4 Adicionando cabeçalhos

A função add_heading() adiciona um parágrafo com um dos estilos de cabeçalho, são passados 2 argumentos para a função, sendo eles o texto que ficará no cabeçalho e um inteiro de 0 a 4, usando o argumento 0 na função o cabeçalho ficará no estilo título, os demais serão sub-cabeçalhos.

Código Exemplo

```
>>> doc = docx.Document()
>>> doc.add_heading('Header 0', 0)
<docx.text.Paragraph object at 0x00000000036CB3C8>
>>> doc.save('headings.docx')
```

2.1.5 Adicionando quebras de linhas e de páginas

Para adicionar uma quebra de linha ao invés de começar um novo parágrafo usa-se o método add_break() no objeto Run anterior à quebra de linha, para adicionar uma quebra de página basta passar o argumento docx.text.WD_BREAK.PAGE

Código Exemplo

```
>>> doc = docx.Document()
>>> doc.add_paragraph('This is on the first page!')
<docx.text.Paragraph object at 0x0000000003785518>
>>> doc.paragraphs[0].runs[0].add_break(docx.text.WD_BREAK.PAGE)
```

```
>>> doc.add_paragraph('This is on the second page!')
<docx.text.Paragraph object at 0x00000000037855F8>
>>> doc.save('twoPage.docx')
```

2.1.6 Adicionando Imagens

O objeto Document conta o método add_picture() que é capaz de adicionar uma imagem ao fim do documento, é possível utilizar a imagem com o tamanho padrão passando como argumento ao método apenas o nome do arquivo de imagem, também é possível especificar a altura e largura da imagem usando as funções width=docx.shared.Inches(n) e height=docx.shared.Cm(n) como argumentos do método.

Código Exemplo

```
>>> doc.add_picture('zophie.png', width=docx.shared.Inches(1),
height=docx.shared.Cm(4))
<docx.shape.InlineShape object at 0x00000000036C7D30>
```

Capítulo 3

Referências

3.0.1 Bibliografia

Trabalho realizado com base no livro Sweigart (2015)

Referências Bibliográficas

Sweigart, A. (2015). *Automate the Boring Stuff with Python: Practical Programming for Total Beginners*. No Starch Press.