Concept

The point of this exercise was to code a script to extract text from a pdf file containing scanned documentatation. Three conditions here:

- input file is available as a pdf link
- output has to be in hOCR format
- process has to be repeatable for similar documents

First of all, I divided problem into smaller parts:

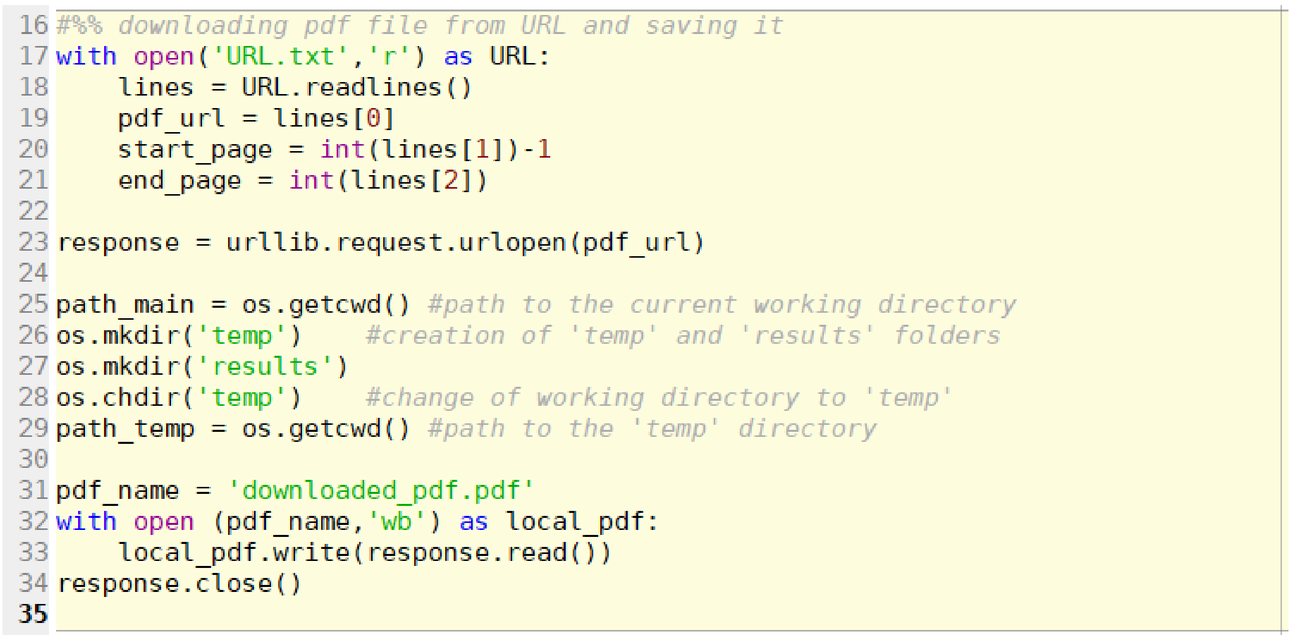
1. downloading pdf file
2. splitting multipages file into a signle-page pdf
3. converting pdf into images
4. binarization
5. applying OCR algorithm
6. saving results as a hOCR files

Realization

I decided to code in Python since it is very flexible programing langueage with well developed libraries.

Point 1

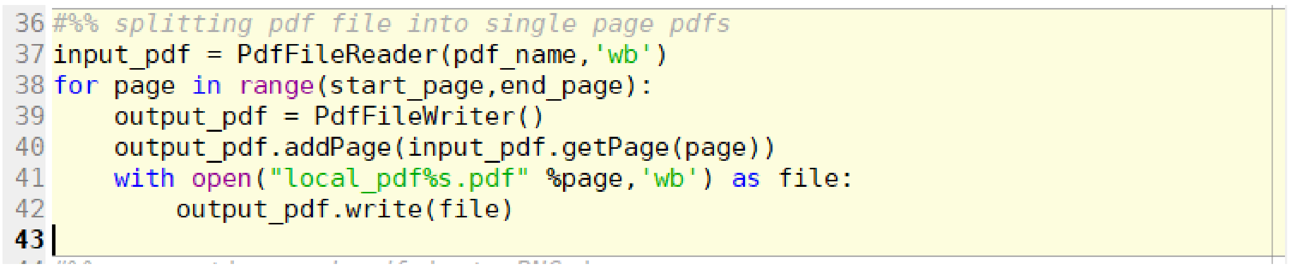
ULR address and page numbers are passed using a txt file. The first line coresponds to URL, second to start a page and the third to last page of the original document. The pdf file is saved in a temporary folder. Rows from 25 ro 29 are responsible for creating path variables and folders where results and temporary files are stored



```
16 #%% downloading pdf file from URL and saving it
17 with open('URL.txt','r') as URL:
18     lines = URL.readlines()
19     pdf_url = lines[0]
20     start_page = int(lines[1])-1
21     end_page = int(lines[2])
22 
23 response = urllib.request.urlopen(pdf_url)
24 
25 path_main = os.getcwd() #path to the current working directory
26 os.mkdir('temp')     #creation of 'temp' and 'results' folders
27 os.mkdir('results')
28 os.chdir('temp')     #change of working directory to 'temp'
29 path_temp = os.getcwd() #path to the 'temp' directory
30 
31 pdf_name = 'downloaded_pdf.pdf'
32 with open (pdf_name,'wb') as local_pdf:
33     local_pdf.write(response.read())
34 response.close()
35 
```

Point 2

Splitting the original file into single line pdfs is performed keeping in mind page numbers specyfing by user in the txt file. Each file is saved in a temp file

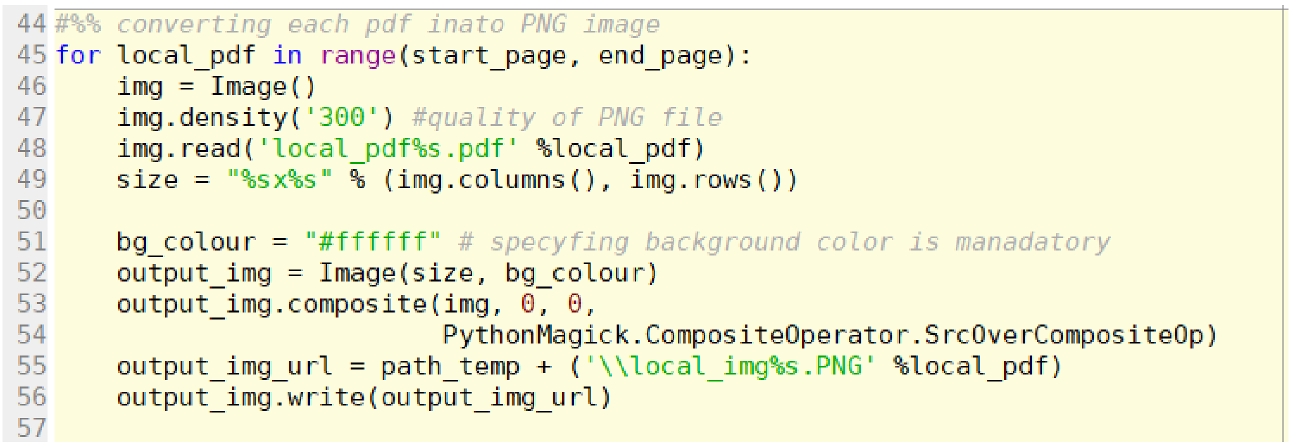


```
36 #%% splitting pdf file into single page pdfs
37 input_pdf = PdfFileReader(pdf_name,'wb')
38 for page in range(start_page,end_page):
39     output_pdf = PdfFileWriter()
40     output_pdf.addPage(input_pdf.getPage(page))
41     with open("local_pdf%s.pdf" %page,'wb') as file:
42         output_pdf.write(file)
43
```

Next steps are executed on per file basis so for loop over files seems to be perfect choice

Point 3

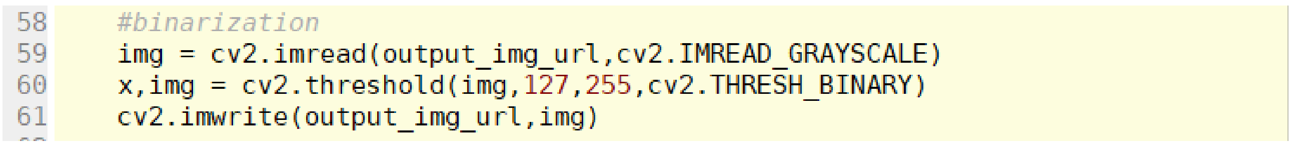
Rows from 46 to 55 are responsible for reading pdf files and saving png images in the 'temp' folder. I chose PNG because it is 'lossess data compression' format



```
44 #%% converting each pdf inato PNG image
45 for local_pdf in range(start_page, end_page):
46     img = Image()
47     img.density('300') #quality of PNG file
48     img.read('local_pdf%s.pdf' %local_pdf)
49     size = "%sx%s" % (img.columns(), img.rows())
50
51     bg_colour = "#ffffff" # specyfing background color is manadatory
52     output_img = Image(size, bg_colour)
53     output_img.composite(img, 0, 0,
54                          PythonMagick.CompositeOperator.SrcOverCompositeOp)
55     output_img_url = path_temp + ('\\local_img%s.PNG' %local_pdf)
56     output_img.write(output_img_url)
57
```

Point 4

Conversion to a grayscale image and then binarization takes place in 58 – 60 rows using global fixed threshold set at 127 level. That level seems to be reasonable considering scans with uniform illumination. The point of binarization is to make images more readible for OCR algorithm. Below code is a part of 'for' loop



```
58     #binarization
59     img = cv2.imread(output_img_url,cv2.IMREAD_GRAYSCALE)
60     x,img = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
61     cv2.imwrite(output_img_url,img)
```

Point 5 & 6

Reading of images is performed by Tesseract engine which is open source tool for optical character recognition. Results are presented in hOCR files (64 row). Output is saved in 'results' folder. Below code is a part of 'for' loop

```
63     #'reading' of image
64     os.chdir(path_main+'\\results')
65     pytesseract.run_tesseract(output_img_url, 'page%s'%local_pdf,   lang='eng',
66                                 boxes=False, config="hocr")
67     os.chdir(path_temp)
68
```

Last step – removing 'temp' folder

Since 'os' library can`t delete non-empty folder, at the very end of 'for' loop, pdf and png files are removed one by one. When 'for' loop is executed, 'temp' folder is removed

```
69     #cleaning temp folder
70     os.remove('local_pdf%s.pdf' %local_pdf)
71     os.remove('local_img%s.PNG' %local_pdf)
72
73 #%% final cleaning
74 os.remove(pdf_name)
75 os.chdir(path_main)
76 os.rmdir('temp')
77
```

Conclusion and accuracy

- Achieved accuracy of nearly 100% is very promising. Applying more case specific image processing techniques can possible improve this result
- Script can be easly converted into exe file and distribute (together with txt file) to users not familiar with programing (Python eniviroment is not required to run program)
- For purposes of greater projects, all content of the script can be converted into class and use a ready to go component

Notes:

- Python script is attached to the email together with hOCR files obtained based on 3 first pages of indicated pdf
- txt and exutable file must be in the same folder
- 'results' folder has to be removed before re-run of the script / .exe file