

Extracting Data I

Collecting Data from PDF


In den meisten Fällen sind die zu verarbeitenden Daten in PDF-Dateien gespeichert. Um Inhalte aus einer pdf-Datei zu extrahieren, verwenden wir die Bibliothek PyPDF2. Diese sollte bereits installiert sein.

```
import PyPDF2 as pypdf

with open('file.pdf', 'rb') as pdf:
    reader = pypdf.PdfFileReader(pdf)
    # Doc-Info ausgeben
    print(reader.documentInfo)
    # Anzahl der Seiten ausgeben
    print(reader.numPages)
```


Essenziell ist die Klasse `PdfFileReader`¹, die jede Menge Properties und Methoden zur Verarbeitung eines PDFs bereitstellt. Details liefert die referenzierte Dokumentation. Als Parameter beim Instanziiieren braucht es ein *File Object*.

Wichtig: Gescannte PDFs können nicht verarbeitet werden – zumindest auf diese Art und Weise nicht!

 2.1 Generieren Sie basierend auf dem Rechnungshofbericht zum Thema „Koordination der Cyber-Sicherheit“² folgenden Output:

```
Author: Rechnungshof
Producer: Adobe PDF Library 15.0
Title: Bericht des Rechnungshofes: Koordination der Cyber-Sicherheit
```

Hinweis: Die Verwendung von `getDocumentInfo` der Klasse `PdfFileReader` ist hierbei sicherlich kein Nachteil.

 2.2 Erweitern Sie die Ausgabe dahingehend, dass ...

- ... der Text der ersten Seite ausgegeben wird.
- ... der gesamte Text ausgegeben wird.
- ... jede Seite des PDFs einer Liste hinzugefügt und am Ende ausgegeben wird.

¹ <https://pythonhosted.org/PyPDF2/PdfFileReader.html>

² https://www.rechnungshof.gv.at/rh/home/home/2022-13_Koordination_Cyber-Sicherheit.pdf

Collecting Data from docx Word Files

Word Files können mithilfe der Bibliothek *python-docx*³ verarbeitet werden. Die prinzipielle Vorgehensweise gleicht jener der PDF-Verarbeitung. Entsprechende Details liefert die referenzierte Literatur. Erwähnenswert ist: Word Files können nicht nur gelesen, sondern auch erstellt werden, wie nachfolgender Quellcode-Auszug verdeutlicht.

```
from docx import Document
from docx.shared import Inches

document = Document()

document.add_heading('Document Title', 0)

p = document.add_paragraph('A plain paragraph having some ')
p.add_run('bold').bold = True
p.add_run(' and some ')
p.add_run('italic.').italic = True

document.add_heading('Heading, level 1', level=1)
document.add_paragraph('Intense quote', style='Intense Quote')

document.add_paragraph(
    'first item in unordered list', style='List Bullet'
)
document.add_paragraph(
    'first item in ordered list', style='List Number'
)

document.add_picture('monty-truth.png', width=Inches(1.25))

records = (
    (3, '101', 'Spam'),
    (7, '422', 'Eggs'),
    (4, '631', 'Spam, spam, eggs, and spam')
)

table = document.add_table(rows=1, cols=3)
hdr_cells = table.rows[0].cells
hdr_cells[0].text = 'Qty'
hdr_cells[1].text = 'Id'
hdr_cells[2].text = 'Desc'
for qty, id, desc in records:
    row_cells = table.add_row().cells
    row_cells[0].text = str(qty)
    row_cells[1].text = id
    row_cells[2].text = desc

document.add_page_break()

document.save('demo.docx')
```

Collecting Data from HTML

Um eine WWW-Ressource – also HTML – zu verarbeiten, benötigen wir die Bibliothek *Beautifulsoup*⁴.

³ <https://python-docx.readthedocs.io/en/latest/>

⁴ <https://pypi.org/project/beautifulsoup4/>

```

# Bibliotheken laden
import urllib.request as urllib2
from bs4 import BeautifulSoup

# Website laden
response = urllib2.urlopen('https://www.htlkrems.ac.at')
html_doc = response.read()

# Das BeautifulSoup Object soup repräsentiert das „geparste“ HTML-Dokument
soup = BeautifulSoup(html_doc, 'html.parser')

# Das „geparste“ HTML-Dokument formatieren, sodass jeder Tag bzw. Textblock
# in einer separaten Zeile ausgegeben wird
strhtml = soup.prettify()

# Ein paar Zeilen ausgeben
print (strhtml[:1000])

```

Output:

```

<!DOCTYPE html>
<html class="no-js" lang="de-DE">
  <head>
    <meta charset="utf-8"/>
    <meta content="width=device-width" name="viewport"/>
    <link href="http://gmpg.org/xfn/11" rel="profile"/>
    <link href="https://www.htlkrems.ac.at/xmlrpc.php" rel="pingback"/>
    <link href="#" rel="stylesheet" title="colors" type="text/css"/>
    <title>
      HTL Krems
    </title>
    <meta content="max-image-preview:large" name="robots">
    <!-- Open Graph Meta Tags generated by Blog2Social 682 -
https://www.blog2social.com -->
    <meta content="HOME" property="og:title">
    <meta content="" property="og:description"/>
    <meta content="https://www.htlkrems.ac.at/" property="og:url"/>
    <!-- Open Graph Meta Tags generated by Blog2Social 682 -
https://www.blog2social.com -->
    <!-- Twitter Card generated by Blog2Social 682 -
https://www.blog2social.com -->
    <meta content="summary" name="twitter:card"/>
    <meta content="HOME" name="twitter:title">
    <meta content="" name="twitter:description"/>

```

Der einfachste Weg, um durch den in `soup` gespeicherten Baum zu navigieren, besteht in der Verwendung des *Tag*-Namens⁵.

```

print(soup.title)
print(soup.title.string)


```


Output:

⁵ <https://www.crummy.com/software/BeautifulSoup/bs4/doc/#navigating-using-tag-names>

```
<title>HTL Krems</title>
```

```
HTL Krems
```

 2.3 Gesucht ist der Text/Inhalt aller *Anchor Tags* (vgl. `<a>`) auf www.htlkrems.ac.at. Geben Sie diesen aus. Hierbei ist die soup-Funktion `find_all()` überaus hilfreich.

 2.4 Ermitteln Sie basierend auf 2.3 die Anzahl der Hyperlinks.