Tutorial

This brief tutorial should give you an introduction and orientation to pikepdf's paradigm and syntax. From there, we refer to you various topics.

Öffnen und Speichern von PDFs

Im Gegensatz zu bekannteren PDF-Bibliotheken verwendet pikepdf ein einzelnes Objekt, um stellen eine PDF-Datei dar, egal ob beim Lesen, Schreiben oder Zusammenführen. Wir haben geschickt benannt Dieses `Pikepdf. PDF`. In dieser Dokumentation ist a eine Klasse, die ermöglicht die Manipulation der PDF, d.h. der "Datei" (ob sie im Speicher oder auf B. ein Dateisystem). `Pdf`

```
from pikepdf import Pdf
new_pdf = Pdf.new()
with Pdf.open('sample.pdf') as pdf:
    pdf.save('output.pdf')
```

Sie können natürlich verwenden, wenn die kurze Klasse Namenskonflikte oder wenn Sie Großbuchstaben bevorzugen. `from pikepdf import Pdf as ...` `from pikepdf import Pdf as PDF`

`pikepdf.open()` ist eine Abkürzung für `pikepdf. pdf.open().`

Die PDF-Klassen-API folgt dem Beispiel der weit verbreiteten Bildbibliothek Pillow. Für Klarheit Es gibt keinen Standardkonstruktor, da die für die Erstellung verwendeten Argumente und Öffnen sind anders. Um eine neue leere PDF-Datei zu erstellen, verwenden Sie not . `Pdf.new()` `Pdf()`

`Pdf.open()` Akzeptiert auch suchbare Streams als Eingabe und `PikePDF. Pdf.save()` akzeptiert Streams als Ausgabe. `pathlib. Pfadobjekte` werden überall dort vollständig unterstützt, wo Pikepdf akzeptiert einen Dateinamen.

Prüfen von Seiten

Das Bearbeiten von Seiten ist für PDFs von grundlegender Bedeutung. pikepdf präsentiert die Seiten in einer PDF-Datei durch das `PikePDF. Pdf.pages` Eigenschaft, die dem Protokoll folgt. Daher beginnen Seitenzahlen bei 0. `list`

Öffnen wir eine einfache PDF-Datei mit vier Seiten.

```
In [1]: from pikepdf import Pdf

In [2]: pdf = Pdf.open('../tests/resources/fourpages.pdf')
```

Wie viele Seiten?

```
In [3]: len(pdf.pages)
Out[3]: 4
```

pikepdf lässt sich in IPython und Jupyters umfangreiche Objekt-APIs integrieren, sodass Sie Anzeigen von PDFs, PDF-Seiten oder Bildern in PDF-Dateien in einem IPython-Fenster oder Jupyter Notizbuch. Dies erleichtert das Testen visueller Veränderungen.

```
In [4]: pdf
Out[4]: « In Jupyter you would see the PDF here »

In [5]: pdf.pages[0]
Out[5]: « In Jupyter you would see an image of the PDF page here »
```

Sie können auch einzelne Seiten untersuchen, die wir im nächsten Abschnitt. Es genügt zu sagen, dass Sie auf Seiten zugreifen können, indem Sie sie indizieren und Schneiden Sie sie.

```
In [6]: pdf.pages[0]
Out[6]: « In Jupyter you would see an image of the PDF page here »
```

> **Anmerkung**
>
> `PikePDF. Pdf.open()` kann fast alle Arten von verschlüsselten PDFs öffnen! Gerade Geben Sie das Schlüsselwortargument an. `password=`

Weitere Informationen zur Dokumentzusammenstellung finden Sie unter PDF-Teilen, Zusammenführen und Zusammenfügen von Dokumenten.

PDF-Wörterbücher

In PDFs ist die Hauptdatenstruktur das **Wörterbuch**, ein Schlüssel-Wert-Daten. -Struktur ähnlich wie Python oder . Der Hauptunterschied besteht darin, dass die Schlüssel nur **Namen** und die Werte nur PDF-Typen sein können, einschließlich andere

Wörterbücher. `dict` `attrdict`

PDF-Wörterbücher werden als `pikepdf dargestellt. Wörterbuchobjekte und Namen` sind vom Typ `Pikepdf. Name`.

```
In [7]: from pikepdf import Pdf

In [8]: example = Pdf.open('../tests/resources/congress.pdf')

In [9]: example.Root  # Show the document's root dictionary
Out[9]:
pikepdf.Dictionary(Type="/Catalog")({
  "/Pages": {
    "/Count": 1,
    "/Kids": [ <Pdf.pages.from_objgen(4,0)> ],
    "/Type": "/Pages"
  },
  "/Type": "/Catalog"
})
```

Page dictionaries

A page in a PDF is just a dictionary with certain required keys that is referenced by the PDF's "page tree". (pikepdf manages the page tree for you, and wraps page dictionaries to provide special functions that help with managing pages.) A `pikepdf.Page` is a wrapper around a PDF page dictionary that provides many useful functions for working on pages.

```
In [10]: from pikepdf import Pdf

In [11]: example = Pdf.open('../tests/resources/congress.pdf')

In [12]: page1 = example.pages[0]

In [13]: obj_page1 = page1.obj

In [14]: obj_page1
Out[14]:
<pikepdf.Dictionary(Type="/Page")({
  "/Contents": pikepdf.Stream(owner=<...>, data=<...>, {
      "/Length": 50
    }),
  "/MediaBox": [ 0, 0, 200, 304 ],
  "/Parent": <reference to /Pages>,
  "/Resources": {
    "/XObject": {
      "/Im0": pikepdf.Stream(owner=<...>, data=<...>, {
          "/BitsPerComponent": 8,
          "/ColorSpace": "/DeviceRGB",
          "/Filter": [ "/DCTDecode" ],
          "/Height": 1520,
          "/Length": 192956,
          "/Subtype": "/Image",
          "/Type": "/XObject",
          "/Width": 1000
        })
    }
  },
  "/Type": "/Page"
})>
```

repr() output

Let's observe the page's output: `repr()`

```
In [15]: repr(page1)
Out[15]: '<pikepdf.Page({\n  "/Contents": pikepdf.Stream(owner=<...>, data=<...>, {\n
"/Length": 50\n    }),\n  "/MediaBox": [ 0, 0, 200, 304 ],\n  "/Parent": <reference to
/Pages>,\n  "/Resources": {\n    "/XObject": {\n      "/Im0": pikepdf.Stream(owner=<...>, data=
<...>, {\n          "/BitsPerComponent": 8,\n          "/ColorSpace": "/DeviceRGB",\n
"/Filter": [ "/DCTDecode" ],\n          "/Height": 1520,\n          "/Length": 192956,\n
"/Subtype": "/Image",\n          "/Type": "/XObject",\n          "/Width": 1000\n        })\n
}\n  },\n  "/Type": "/Page"\n})>'
```

The angle brackets in the output indicate that this object cannot be constructed with a Python expression because it contains a reference. When angle brackets are omitted from the of a pikepdf object, then the object can be replicated with a Python expression, such as . Pages typically have indirect references to themselves and other pages, so they cannot be represented as an expression. `repr()` `eval(repr(x)) == x`

Item and attribute notation

Dictionary keys may be looked up using attributes () or keys (). `page1.Type` `page1['/Type']`

```
In [16]: page1.Type      # preferred notation for standard PDF names
Out[16]: pikepdf.Name("/Page")

In [17]: page1['/Type']  # also works
Out[17]: pikepdf.Name("/Page")
```

Gemäß der Konvention verwendet pikepdf Attributnotation für Standardnamen (die Namen die normalerweise Teil eines Wörterbuchs sind, gemäß dem PDF 1.7 Referenzhandbuch), und Elementnotation für Namen, die möglicherweise nicht immer angezeigt werden. Zum Beispiel die Bilder zu einer Seite gehören immer unter aber die Namen der Bilder werden willkürlich von der Software ausgewählt, die das PDF generiert (, in diesem Fall). (Wenn sie als Zeichenfolgen ausgedrückt werden, beginnen Namen mit .) `page.Resources.XObject` `/Im0` `/`

```
In [18]: page1.Resources.XObject['/Im0']
```

Die Artikelnotation wäre hier ziemlich umständlich: (nicht empfohlen). `['/Resources']['/XObject]['/Im0']`

Die Attributnotation ist praktisch, aber nicht robust, wenn Elemente fehlen. Für Elemente, die nicht immer vorhanden sind, können Sie verwenden, die sich wie im Kernpython verhalten. Eine Bibliothek wie glom kann bei der Arbeit mit komplexen strukturierte Daten, die nicht immer vorhanden sind. `.get()` `dict.get()`

(Im Moment werden wir beiseite legen, wofür eine Seite gedacht ist. Weitere Informationen finden Sie unter Arbeiten mit Seiten.) `Resources.XObject`

Löschen von Seiten

Das Entfernen von Seiten ist auch einfach.

```
In [19]: del pdf.pages[1:3]  # Remove pages 2-3 labeled "second page" and "third page"
```

```
In [20]: len(pdf.pages)
Out[20]: 2
```

Speichern von Änderungen

Natürlich können Sie Ihre Änderungen mit `pikepdf speichern. pdf.save().` kann eine `Pathlib sein. Weg`, den wir überall akzeptieren. `filename`

```
In [21]: pdf.save('output.pdf')
```

Hecht retten.¶

Sie können eine Datei mehrmals speichern und nach Ersparnis. Sie können beispielsweise eine unverschlüsselte Version des Dokuments erstellen und dann Wenden Sie ein Wasserzeichen an, und erstellen Sie eine verschlüsselte Version.

> **Anmerkung**
>
> Sie dürfen die Eingabedatei (oder das Python-Objekt, das die Eingabedatei bereitstellt) nicht überschreiben. B. Daten) beim Speichern oder zu einem anderen Zeitpunkt. Pikepdf geht davon aus, dass es Exklusiver Zugriff auf die Eingabedatei oder Eingabedaten, denen Sie sie geben, bis sie aufgerufen werden. `pdf.close()`

Speichern sicherer PDFs

Um eine verschlüsselte (kennwortgeschützte) PDF-Datei zu speichern, verwenden Sie ein Objekt, um die Verschlüsselungseinstellungen festzulegen. Standardmäßig wählt pikepdf die Option stärkster Sicherheitshandler und -algorithmus (AES-256), ermöglicht jedoch vollen Zugriff auf Ändern Sie den Dateiinhalt. Ein `Pikepdf. Das Permissions-Objekt` kann verwendet werden, um Geben Sie Einschränkungen an. `pikepdf.Encryption`

```
In [22]: no_extracting = pikepdf.Permissions(extract=False)

In [23]: pdf.save('encrypted.pdf', encryption=pikepdf.Encryption(
   ....:     user="user password", owner="owner password", allow=no_extracting
   ....: ))
   ....:
```

Weitere Informationen finden Sie in unserer Sicherheitsdokumentation zu Benutzer-/Besitzerkennwörter und PDF-Berechtigungen.

Ausführen von QPDF über Jobs

pikepdf kann auf alle Funktionen des qpdf-Befehlszeilenprogramms zugreifen und kann sogar qpdf-ähnliche Befehlszeilen ausführen.

```
In [24]: from pikepdf import Job

In [25]: Job(['pikepdf', '--check', '../tests/resources/fourpages.pdf'])
```

Sie können Jobs auch in QPDF Job JSON angeben:

```
In [26]: job_json = {'inputFile': '../tests/resources/fourpages.pdf', 'check': ''}

In [27]: Job(job_json).run()
```

Nächste Schritte

Werfen Sie einen Blick auf pikepdf-Themen, die Sie interessieren, oder springen Sie zu unserer detaillierten API Referenz...