## Vorwort

Das folgende Dokument wurde als PDF-Export eines Jupyter-Notebook-Dokuments (https://jupyter.org/) generiert. Diese PDF-Version enthält denselben Inhalt, aber das Code-Highlighting ist etwas unübersichtlicher und der Code ist nicht direkt ausführbar. Wenn Sie stattdessen direkt das IPYNB-Dokument öffnen möchten, benötigen Sie eine Jupyter-Notebook-Installation (https://jupyter.org/install.html). Diese installieren Sie, falls Sie Python bereits installiert haben, mit dem Befehl > pip install notebook. Das Programm starten können Sie anschließend mit > jupyter notebook.

## Einleitung

In diesem Notebook finden Sie erklärten Code, mit dem Sie alle Ressourcen eines Zeitungsunernehmens herunterladen können, die mit Visual Library gehostet werden.

Diese dritte Version erweitert die zweite Version um die Möglichkeit, sehr detailliert festzulegen, wie die Ordner und die heruntergeladenen Dateien heißen sollen. Dies kann unter anderem für die anschließende Weiterverarbeitung mit anderen Programmen hilfreich sein.

Das hier erläuterte Beispiel ist das Zeitungsunternehmen Augsburger Postzeitung: https://visuallibrary.net/dps/periodical/titleinfo/436884.

Um diesem Notebook folgen zu können, ist ein Blick in die erste und zweite Version wahrscheinlich hilfreich, da dort das prinzipielle Vorgehen und einige verwendete Techniken genauer erklärt werden.

In dieser Version liegt der Fokus darauf, die Optionen zur Namenskonfiguration zu erklären; nicht darauf, den Code zu erklären, der dies bewerkstelligt.

## Idee

Der Ansatz ist derselbe wie in der Basisversion: Über die OAI/METS-Dokumente wird von der höchsten Hierarchie-Ebene (dem Zeitungsunternehmen) subsequent auf die Kinder der jeweils tieferen Ebene zugegriffen. Daneben werden mit XPath-Abfragen detaillierte Metadaten aus den MODS-Elementen (http://dfg-viewer.de/fileadmin/groups/dfgviewer/MODS-Anwendungsprofil_2.3.1.pdf) extrahiert, die in die METS-Dokumente eingebettet sind.

Um die Informationen aus den MODS-Elementen für die Speicherung zu verwenden, benutzen wir Funktionen, um jeden einzelnen logischen Teil, also Zeitungsunternehmen - Zeitung - Jahrgang - Ausgabe - Seite, in die gewünschte Namensform zu bringen. Die Namen dieser logischen Einheiten können dann an zwei verschiedenen Stellen genutzt werden: einerseits bei der Erstellung der Ordnernamen, in denen die einzelnen Seiten gespeichert werden sollen; andererseits in den Dateinamen der Seiten.

Die üblichwerweise vorhandenen Ressourcen sind ein Bild von jeder Zeitungsseite in unterschiedlicher Auflösung und ein PDF-Dokument von jeder Ausgabe. Daneben gibt es häufig Volltexte von einzelnen Seiten, meistens im ALTO-Format, manchmal auch (zusätzlich) im HTML-Format.

## Code

Wir erledigen zunächst die notwendigen Importe und definieren die XML-Namespaces.

In [ ]:
```
# %pip install lxml
# %pip install requests
```

In [ ]:
```
from io import StringIO
from mimetypes import guess_extension
from os import makedirs, path, rmdir
from datetime import date
from lxml import etree
import requests


ns_map = {'xmlns': "http://www.openarchives.org/OAI/2.0/",
          'mets': "http://www.loc.gov/METS/",
          'xlink': "http://www.w3.org/1999/xlink",
          'mods': "http://www.loc.gov/mods/v3"}
```

Wir geben die URL des Zeitungsunternehmens an.

In [ ]:
```
newspaper_series_url = 'https://visuallibrary.net/dps/oai/?verb=GetRecord&metadataPrefix=r
```

Wir legen fest, welche Formate wir herunterladen wollen. Wenn mehr als eines der Formate 'Max', 'Min', 'Default', 'Thumbs' benutzt wird, wird das letzte davon die vorherigen überschreiben, weil sie namensgleiche JPEG-Dateien erzeugen.

In [ ]:
```
# formats = ['Fulltext', 'Min', 'Thumbs', 'Default', 'Max', 'Download']
formats = ['Fulltext', 'Max', 'Download']
```

## Namenskonfiguration

Hier kann das Download-Verzeichnis festgelegt werden. In dieses und seine noch festzulegenden Unterverzeichnisse werden alle Ressourcen heruntergeladen. Es kann ein relativer Pfad sein, d.h. im Ordner, wo dieses Notebook gespeichert ist, wird ein neuer Ordner dieses Namens erzeugt, oder ein absoluter Pfad, d.h. ausgehend vom Laufwerk wird der genaue Pfad angegeben.

In [ ]:
```
base = "downloads"
```

Nun kann die Ordnerstruktur festgelegt werden, in der die einzelnen Seiten gespeichert werden sollen. Die sechs Bestandteile 'base', 'newspaper_series', 'newspaper', 'year', 'issue' und 'fmt' können beliebig arangiert und/oder ausgelassen werden. Die Ordnernamen werden gemäß dieser Liste von links nach rechts gebildet. Dadurch kann die Feinheit der Ordnergliederung beliebig kontrolliert werden. Im Wesentlichen gibt es hier zwei Entscheidungen zu treffen. Erstens kann festgelegt werden, wie tief die Ordnerstruktur die logische Zugehörigkeitsstruktur einer Seite nachbilden soll. Zweitens kann festgelegt werden, in welcher Tiefe die Aufspaltung in verschiedene Formate stattfinden soll. Einige Beispiele:

A) Alle Ressourcen, die zu einer Zeitung gehören, sollen in dasselbe Verzeichnis heruntergeladen werden. Diese Verzeichnisse sollen einfach (für jede Zeitung) im aktuellen Ordner erstellt werden: </br> folder_structure = ['newspaper']

B) Analog zu A, aber die verschiedenen Formate sollen für jede Zeitung in getrennten Ordnern gespeichert werden: </br> folder_structure = ['newspaper', 'fmt']

C) Im aktuellen Verzeichnis soll ein neuer Ordner angelegt werden. Es soll zuerst nach Formaten unterschieden werden, dann für jeden Jahrgang ein Ordner angelegt werden, in diesem soll nach Zeitungen separiert

gespeichert werden, und zuletzt in jedem Zeitungsordner jede Ausgabe in einem eigenen Ordner: </br>
folder_structure = ['base', 'fmt', 'year', 'newspaper', 'issue']

In [ ]:
```
folder_structure = ['base', 'newspaper_series', 'newspaper', 'year', 'issue', 'fmt']
```

Hierbei ist zu beachten, dass es insbesondere beim Speichern in einer wenig verästelten Ordnerstruktur schnell zu Namensdopplungen kommen kann. Wenn beispielsweise wie in B nur nach Zeitung und Format getrennt wird, und die Namen jeder einzelnen Seite dem Muster 'Seite X' folgen, werden die einzelnen Seiten jeder Ausgabe jedes Jahrgangs gleich heißen und die zuletzt heruntergeladene Seite die vorherigen dementsprechend überschreiben.

Um das zu vermeiden, kann neben der Ordnerstruktur auch der Name jeder einzelnen Datei sehr genau festgelegt werden. Alle Informationen über die Zugehörigkeit zu höheren Einheiten (Unternehmen, Zeitung, Jahrgang, Ausgabe) können optional im Dateinamen enthalten sein, außerdem noch die Seitenanzahl, die automatisch ermittelt wird, und die eindeutige Visual Library Speicher-ID der Seite.

Der Name einer Einzelseite wird einfach als String festgelegt, wobei Referenzen zu den Zugehörigkeitsebenen genutzt werden können, welche dann durch die konkrete Information ersetzt werden. Die anderen Bestandteile des Strings werden nicht verändert, zum Beispiel könnte man die Seiten mit 'Seite PAGE_NO' benennen, wobei PAGE_NO durch die konkrete Seitennummer ersetzt würde, sodass die Dateien dann "Seite 1", "Seite 2", usw. hießen.

Sollte man wie in B jedoch nur nach Zeitung und Format trennen, wäre es wahrscheinlich sinnvoll, die unterscheidenden Informationen wie Jahrgang oder Ausgabe in den Namen der Einzelseiten zu verwenden. Dann könnte der Variable page_name_format z.B. so aussehen: 'YEAR_NAME ISSUE_NAME Seite PAGE_NO'.

Die PDF-Dokumente einer ganzen Ausgabe werden der Einfachheit halber wie die Namen von Einzelseiten behandelt. PAGE_ID ist die eindeutige ID des PDFs. PAGE_NO wird als Umfang angegeben, beispielsweise 1-4.

In [ ]:
```
# parts that will be replaced: NEWSPAPER_SERIES_NAME, NEWSPAPER_NAME, YEAR_NAME, ISSUE_NAM
page_name_format = 'Seite PAGE_NO'
```

Die einzelnen Bestandteile, die wir bei der Benennung von Ordnern und Dateien benutzt haben, können selbst ein Stück weit angepasst werden.

Für den Jahrgang wird aus dem MODS das Jahr extrahiert, auf dieses kann im Namen für den Jahrgang mit dem Substring '%Y' zugegriffen werden. (Erläutert wird dies hier: https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes.) Dies ist auch der Defaultwert. Es wäre aber beispielsweise auch möglich, den Wert 'Jahrgang %Y' zu nutzen, dann würde, sofern die Information in der Ordnerstruktur und/oder für die Dateinamen verwendet wird, der Teil YEAR_NAME durch 'Jahrgang YYYY' ersetzt.

In [ ]:
```
year_name_format = '%Y'
```

Analog dazu kann der Name der Ausgabe angepasst werden. Außerdem kann festgelegt werden, ob neben dem Erscheinungsdatum weitere (verfügbare) Informationen über die Ausgabe zum Namen hinzugefügt werden sollen, z.B. die laufende Nummer im Jahr oder Zusätze wie 'Vorabendblatt', die z.B. verschiedene Ausgaben am selben Tag differenzieren können.

In [ ]:
```
issue_name_format = '%Y-%m-%d'
issue_name_granularity = True
```

Die Namen für das Zeitungsunternehmen und für die einzelnen Zeitungen werden automatisch aus den MODS-Informationen konstruiert, es ist allerdings auch möglich, eigene Namen zu verwenden. Für das Zeitungsunternehmen wird einfach ein String übergeben, für die Zeitungen ein Dictionary, in dem für jede Zeitungs-ID der Name festgelegt werden kann.

In [ ]:
```
newspaper_series_name = ""
newspaper_names = {
    'newspaper_id': 'name',
}
```

Zuletzt kann spezifiziert werden, hinter welche Namensteile zusätzlich die eindeutige Visual-Library-ID angehangen werden soll. Dies kann nützlich sein, um die einzelnen Einheiten (online) schnell wiederzufinden. Außerdem kann es in seltenen Fälle vorkommen, dass die METS-Informationen stellenweise ungenau sind, und z.B. einer Ausgabe fälschlicherweise dasselbe Datum wie einer anderen Ausgabe zugewiesen wird. Dann könnten die ansonsten gleichnamigen Seiten der zwei Ausgaben durch die angehangene VL-ID unterschieden werden und eine Überschreibung würde verhindert.

Für jede Strukturebene kann individuell entschieden werden, ob die ID angegangen werden soll. Sie wird angehangen für alle Ebenenen, die in der Liste enthalten sind.

In [ ]:
```
# append_VL_ID = ['newspaper_series', 'newspaper', 'issue', 'year', 'page']
append_VL_ID = ['page']
```

Die Möglichkeiten, Namen für Ordner und Dateien festzulegen, sind also sehr vielfältig. Wahrscheinlich ist es sinnvoll, zunächst auszuprobieren, ob die Ressourcen tatsächlich mit den gewünschten Namen gespeichert werden, bevor der mitunter lange dauernde Download aller Ressourcen durchgeführt wird.

Falls Sie unter Windows arbeiten, sollten Sie außerdem beachten, dass die maximale Pfadlänge unter Standardeinstellungen 260 Zeichen beträgt. Mit diesem Skript kann es durchaus passieren, dass Sie diese Länge überschreiten, da Sie einerseits eine recht tief verzweigte Pfadstruktur erzeugen können und andererseits die in VL verwendeten Namen für Zeitungen sehr lang sein können. Dieses Problem ist in einer weiteren Version gelöst, die von der Eingabeeinforderung aus aufgerufen werden kann, und noch einige weitere zusätzliche Funktionalitäten umfasst.

## Funktionsdefinitionen

In [ ]:
```
s = requests.Session()
s.headers.update({'User-Agent': 'METS_Downloader'})

# get root node as lxml node from a url
get_root = lambda url: etree.parse(StringIO(s.get(url).text)).getroot()

get_newspapers_from_newspaper_series = lambda newspaper_series_url: \
    get_root(newspaper_series_url).xpath('//mets:structMap//@xlink:href', namespaces=ns_ma

get_years_from_newspaper = lambda newspaper_url: \
    get_root(newspaper_url).xpath('//mets:structMap//@xlink:href', namespaces=ns_map)

get_issues_from_year = lambda year_url : \
    get_root(year_url).xpath('//mets:structMap//*[@TYPE="day"]//@xlink:href', namespaces=n

def get_pages_from_issue(url, fmt):
    """Get pages from an issue-url, provide format which should be extracted, return list
    root = get_root(url)
    page_links = root.xpath(f'//mets:fileGrp[@USE="{fmt.upper()}"]//@xlink:href', namespac
```

```
    return page_links

def get_newspaper_series_name(newspaper_series_url, newspaper_series_name, append_VL_ID):
    if newspaper_series_name:
        newspaper_series_name_formatted = newspaper_series_name
    else:
        root = get_root(newspaper_series_url)
        parts = root.xpath(f'//mets:dmdSec[@ID="md{get_id(newspaper_series_url)}"]//mods:t
        newspaper_series_name_formatted = ""
        for part in parts:
            newspaper_series_name_formatted += f'{part.text}. '

        newspaper_series_name_formatted = newspaper_series_name_formatted.rstrip('. ')

    if 'newspaper_series' in append_VL_ID:
        newspaper_series_name_formatted += f'.{get_id(newspaper_series_url)}'

    return newspaper_series_name_formatted

def get_newspaper_name(newspaper_url, newspaper_names, append_VL_ID):
    if get_id(newspaper_url) in newspaper_names:
        newspapper_name_formatted = newspaper_names[get_id(newspaper_url)]
    else:
        root = get_root(newspaper_url)
        parts = root.xpath(f'//mets:dmdSec[@ID="md{get_id(newspaper_url)}"]//mods:titleIn
        newspapper_name_formatted = ""
        for part in parts:
            newspapper_name_formatted += f'{part.text}. '

        newspapper_name_formatted = newspapper_name_formatted.rstrip('. ')

    if 'newspaper' in append_VL_ID:
        newspapper_name_formatted += f'.{get_id(newspaper_url)}'

    return newspapper_name_formatted

def get_year_name(year_url, year_date_format, append_VL_ID):
    root = get_root(year_url)
    year_date_string = root.xpath(f'//mets:dmdSec[@ID="md{get_id(year_url)}"]//mods:date',
    year_date = date(int(year_date_string), 1, 1)
    year_name_formatted = year_date.strftime(year_date_format)

    if 'year' in append_VL_ID:
        year_name_formatted += f'.{get_id(year_url)}'

    return year_name_formatted

def get_issue_name(issue_url, issue_date_format, issue_name_granularity, append_VL_ID):
    root = get_root(issue_url)
    issue_date_string = root.xpath(f'//mets:dmdSec[@ID="md{get_id(issue_url)}"]//mods:date
    issue_date = date.fromisoformat(issue_date_string)
    issue_name_formatted = issue_date.strftime(issue_date_format)

    # if issue_name_granularity == True, append detail information like number, title abou
    if issue_name_granularity:
        detail = root.xpath(f'//mets:dmdSec[@ID="md{get_id(issue_url)}"]//mods:detail[@typ
        if detail:
            for child in detail[0]:
                if child.text != None:
                    issue_name_formatted += f' {child.text}.'

            issue_name_formatted = issue_name_formatted.rstrip('.')

    if 'issue' in append_VL_ID:
        issue_name_formatted += f'.{get_id(issue_url)}'
```

```
    return issue_name_formatted

get_id = lambda string: string.split('=')[-1]

def get_formatted_page_name(page_id, page_name_format, newspaper_series_url, newspaper_ser
                            newspaper_names, year_url, year_date_format, issue_url, issue_
    """Get file name for a page according to specified format. Use issue to fetch necessar
    issue_root = get_root(issue_url)

    if fmt.lower() == 'download':
        pages = issue_root.xpath(f'//mets:structMap[@TYPE="PHYSICAL"]/mets:div', namespace
        first_page = pages[0].attrib['ORDER']
        last_page = pages[-1].attrib['ORDER']
        page_no = f'{first_page}-{last_page}'
    else:
        page_no = issue_root.xpath(f'//mets:div[@ID="phys{page_id}"]/@ORDER', namespaces=n

    formatted_page_name = page_name_format.replace('PAGE_NO', page_no)

    if 'PAGE_ID' in page_name_format:
        formatted_page_name = formatted_page_name.replace('PAGE_ID', page_id)
    if 'ISSUE_NAME' in page_name_format:
        issue_name = get_issue_name(issue_url, issue_date_format, issue_name_granularity,
        formatted_page_name = formatted_page_name.replace('ISSUE_NAME', issue_name)
    if 'YEAR_NAME' in page_name_format:
        year_name = get_year_name(year_url, year_date_format, append_VL_ID)
        formatted_page_name = formatted_page_name.replace('YEAR_NAME', year_name)
    if 'NEWSPAPER_NAME' in page_name_format:
        newspaper_name = get_newspaper_name(newspaper_url, newspaper_names, append_VL_ID)
        formatted_page_name = formatted_page_name.replace('NEWSPAPER_NAME', newspaper_name
    if 'NEWSPAPER_SERIES_NAME' in page_name_format:
        newspaper_series_name = get_newspaper_series_name(newspaper_series_url, newspaper_
        formatted_page_name = formatted_page_name.replace('NEWSPAPER_SERIES_NAME', newspap

    if 'page' in append_VL_ID:
        formatted_page_name += f'.{page_id}'

    return formatted_page_name


def create_download_path(newspaper_series_url, newspaper_series_name, newspaper_url, newsp
                         year_url, year_date_format, issue_url, issue_date_format, issue_n
                         base, append_VL_ID):
    newspaper_series = get_newspaper_series_name(newspaper_series_url, newspaper_series_na
    newspaper = get_newspaper_name(newspaper_url, newspaper_names, append_VL_ID)
    year = get_year_name(year_url, year_date_format, append_VL_ID)
    issue = get_issue_name(issue_url, issue_date_format, issue_name_granularity, append_VL

    download_path = ""
    for element in folder_structure:
        download_path += f'{eval(element)}/'
    download_path = download_path.rstrip('/')

    try:
        makedirs(f'{download_path}')
    except FileExistsError:
        pass

    return download_path


def download_page(page_link, download_path, page_name_format, newspaper_series_url, newspa
                                            newspaper_url, newspaper_names, year_url, year
                                            issue_name_granularity, fmt, append_VL_ID):
    """Extract name and filetype from link to newspaper page. Then save at specified folde
```

```
    r = s.get(page_link)
    page_id = page_link.split('/')[-1]

    # get page name according to desired format
    if page_name_format == 'PAGE_ID':
        page_name = page_id
    else:
        page_name = get_formatted_page_name(page_id, page_name_format, newspaper_series_ur
                                            newspaper_url, newspaper_names, year_url, year
                                            issue_url, issue_date_format, issue_name_granu

    # get rid of encoding if provided in r.headers; if not provided: split still works, th
    filetype = guess_extension(r.headers['Content-Type'].split(';')[0])
    filename = page_name + filetype

    # throw warning if file will be overwritten
    if path.isfile(f'{download_path}/{filename}'):
        print(f'Warning: {filename} already existed and was overwritten')

    # save file
    with open(f'{download_path}/{filename}', 'wb') as file:
        file.write(r.content)
```

## Download

In [ ]:

```
for newspaper_url in get_newspapers_from_newspaper_series(newspaper_series_url):
    for year_url in get_years_from_newspaper(newspaper_url)[:2]:
        for issue_url in get_issues_from_year(year_url):
            for fmt in formats:
                download_path = create_download_path(newspaper_series_url, newspaper_serie
                                                     newspaper_names, year_url, year_name_
                                                     issue_name_granularity, fmt, folder_s
                for page_link in get_pages_from_issue(issue_url, fmt=fmt):
                    download_page(page_link, download_path, page_name_format, newspaper_se
                                  newspaper_url, newspaper_names, year_url, year_name_forr
                                  fmt, append_VL_ID)

                # Remove empty directories. This can happen if one format (e.g. Fulltext)
                try:
                    rmdir(download_path)
                except Exception:
                    pass
```