Introduction to Python and PyMARC

Session II: Using PyMARC

Lauren Magnuson California State University, San Marcos

Hosted by ALCTS: Association for Library Collections & Technical Services
Quick Recap

Last week, we:

- Learned how to install Python and PyMARC
- Learned how to run scripts using a command line interface (PowerShell, Terminal)
- Went over the GitHub repository of sample scripts
Session II Overview

- Getting fields from MARC files
  - Handling missing fields
- Manipulating fields and string data
- Looping through records and fields
- Writing data from MARC records to CSV
- Writing data from CSV to MARC
Getting fields from MARC records

```python
from pymarc import MARCReader

# print all OCLC numbers (035|a)
with open('marc.mrc', 'rb') as fh:
    reader = MARCReader(fh)
    for record in reader:
        print(record['035']['a'])
```

Note: What is rb? It means “read binary” - [https://docs.python.org/3/library/functions.html#open](https://docs.python.org/3/library/functions.html#open)
What if fields are missing in one of the records?

If we manually remove all 035 fields from the record (see marc_missing035.mrc)

TypeError: 'NoneType' object has no attribute '__getitem__'
What if fields are missing in one of the records?

TypeError: 'NoneType' object has no attribute '__getitem__'

Translation: You told Python to look for an object (the MARC 035 field) but there wasn’t one there - so Python didn’t know how to get the field you asked for. The object has “NoneType”.

How to handle: check for presence of fields with conditional:

```python
if record['035'] is not None:
```
What if fields are missing in one of the records?

```python
from pymarc import MARCReader

# print all OCLC numbers (035|a)
with open('marc_missing035.mrc', 'rb') as fh:
    reader = MARCReader(fh)
    for record in reader:
        if record['035'] is not None:
            print(record['035']['a'])
```

`oclc1_missing.py`
Be sure to check for missing subfields!

```python
from pymarc import MARCReader

# print all OCLC numbers (035|a)
with open('marc_missing035.mrc', 'rb') as fh:
    reader = MARCReader(fh)
    for record in reader:
        if record['035'] is not None:
            if record['035']['a'] is not None:
                print(record['035']['a'])
```

`oclc1_missing.py`
Manipulating fields and string data

#remove OCN if present

```python
if oclc_number.find("ocn") >= 0:
    oclc_number = oclc_number.replace('ocn', '"
else:
    oclc_number = oclc_number
```

Note: Remember to keep your if / else statements aligned on the left, and use consistent indentation!
Manipulating fields and string data - Activity 1

How can you adjust the oclc2 script to get rid of the ocm prefix in two of the items?

```python
#remove OCM if present - needs work!

if oclc_number.find("xyz") >= 0:
    oclc_number = oclc_number.replace('xyz', '')
else:
    oclc_number = oclc_number
```
How can you adjust the oclc2 script to get rid of the ocm prefix in two of the items?

```python
#remove OCM if present - needs work!
if oclc_number.find("ocm") >= 0:
    oclc_number = oclc_number.replace('ocm', '')
else:
    oclc_number = oclc_number
```

Note: have more if conditions? Use if/elif/else:
https://docs.python.org/3/tutorial/controlflow.html
Manipulating fields and string data

```python
if oclc_number.find("ocn") >= 0:
```

Why >=0? The `find` method evaluates a string and returns the index position of the match if a match is found.

- False [no match found] = -1
- True [match found] = 0+

You can read the above code as if the following exchange occurred:

You: Hey Python, take a look at the OCLC number field. Do you see ‘ocn’ in the string?
Python: 15 [translation: yup I found it and it starts at position 15 of the string]
find vs. startswith (and other methods)

#remove OCN if present

```python
if oclc_number.startswith("ocn"):
    oclc_number = oclc_number.replace('ocn', '')
else:
    oclc_number = oclc_number
```

Note: String methods reference: [https://docs.python.org/3/library/stdtypes.html#string-methods](https://docs.python.org/3/library/stdtypes.html#string-methods)
Regular Expressions

```python
if oclc_number.find("01cals_network") >= 0:
    oclc_number = oclc_number.replace('-01cals_network', '')
    oclc_number = re.sub("[^0-9]", ", oclc_number)
else:
    #remove everything that remains except the numbers
    oclc_number = re.sub("[^0-9]", ", record['035']['a'])
```

Output:

```
318875078
668941926
```
What happens if we switch things up?

```python
if oclc_number.find("01cals_network") >= 0:
    oclc_number = re.sub("[^0-9]", "", oclc_number)
    oclc_number = oclc_number.replace('-01cals_network', '"
else:
    # remove everything that remains except the numbers
    oclc_number = re.sub("[^0-9]", "", record['035'][a'])
```

If we put our regex before our find/replace, some numbers are left in (not what we want!)

Output:

```
318875078 01
668941926 01
```

Regexes are very powerful, but follow your instructions in the order you give them.
Looping through records

A loop enables you to run Python code multiple times over a set of objects - in this case, MARC records and fields.

PyMARC knows how to locate individual records within a single MARC file. To tell Python to loop through each record.

```python
for record in reader:
```
Looping through fields

```python
# print all ISBNs (020|a)
with open('marc.mrc', 'rb') as fh:
    reader = MARCReader(fh)
    # loop 1
    for record in reader:
        # loop 2
        for f in record.get_fields('020'):
            # make sure 020|a exists
            if f['a'] is not None:
                isbn = (f['a'])
```

For each record....

For each field....

`isbn-gobi.py`
Looping through fields

```python
# print all ISBNs (020|a)
with open('marc.mrc', 'rb') as fh:
    reader = MARCReader(fh)
    # loop 1
    for record in reader:
        # loop 2
        for f in record.get_fields('020'):
            # make sure 020|a exists
            if f['a'] is not None:
                isbn = (f['a'])
                # Only include numbers
                isbn = re.sub("[^X0-9]", "", isbn)
                # write to CSV file
                csv_out.writerow([isbn, account])
```

Note the indentations and alignment - when looping and using conditionals, indentation is extremely important
Writing data from MARC To CSV

#create a CSV file - note tab delimiter would be '/t'
csv_out = csv.writer(open('output.csv', 'w'), delimiter = ',', quotechar = '"', quoting = csv.QUOTE_ALL)

Note: Quoting is used by Excel or Spreadsheet readers to separate fields that might contain delimiters within the data itself that aren’t meant to be used as delimiters; e.g., commas that might appear within the title of a book.
Writing data from MARC to CSV

#write a header row in your CSV file

```python
csv_out.writerow(['title', 'oclc_number'])
```

....

```python
for record in reader:
    oclc_number = title = ''
```

Declare your variables as blank before you assign them. Otherwise you might get a ‘list index out of range’ error if the object is not found for a particular record.

https://stackoverflow.com/questions/1098643/indexerror-list-index-out-of-range-and-python

```python
oclc-regex-csv.py
```
Writing data from CSV to MARC

Use case:

Another library or a large donation of materials needs to be loaded into your catalog, but the existing data about items can only be extracted as CSV or XML. You’ll need to:

- Turn these items into MARC records
- Clean up OCLC numbers
- Add $9 LOCAL to local titles in records

![records.csv](image)
Writing data from CSV to MARC

```python
import csv
import re
import pymarc
import sys
```

Sys is a module that enables you to pass arguments on the command line. In this script, rather than hardcoding the filename we're parsing, we'll set up the script so that we can swap out the name of our input file each time we run the script.
Writing data from CSV To MARC

# define a function

def csvmarcwriter(file):

# Open your CSV File

with open(file) as fh:
    itemread = csv.reader(fh)
    itemlist = list(itemread)
Writing data from CSV to MARC

# create an output file for the created MARC records (wb means 'write binary')

    outputfile = open('writer.mrc', 'wb')
Writing data from CSV to MARC

#iterate through each row of the CSV file
for row in itemlist[1:]:
    #declare PyMARC record object
    item_load = pymarc.Record(to_unicode=True, force_utf8=True)

    #define data fields in CSV file
    ocn = row[0]
    barcode = row[1]
    ltitle = row[2]
Writing data from CSV to MARC

#Clean up OCLC numbers with regular expression

```
import re

ocn = re.sub("[^0-9]", "", ocn)
```
Writing data from CSV To MARC

#write data to field variables

```python
field_001 = pymarc.Field(tag='001', data=ocn)
field_974 = pymarc.Field(
    tag='974',
    indicators=[' ',' '],
    subfields=['a', ltitle, '9', "LOCAL"],
)
field_949 = pymarc.Field(
    tag='949',
    indicators=[' ',' '],
    subfields=['a', barcode]
)
```

Note repeating subfields to create field like this:

```
974 __ $aPython Collection$9LOCAL
```
Writing data from CSV to MARC

```python
#add field variables to PyMARC record object
    item_load.add_ordered_field(field_001)
    item_load.add_ordered_field(field_974)
    item_load.add_ordered_field(field_949)

#Create output file
    outputfile.write(item_load.as_marc())
```

`marc_writer.py`
Close the output file and call the function

```python
# close the output file
outputfile.close()

file = sys.argv[1]  # There's sys again
csvmarcwriter(file)

# To call the function, try: python marc_writer.py records.csv
```

This argument is your input CSV file.
MARCWriter Output

=LDR 00118 a2200061 4500
=001 668941926
=949 \$a9999999999999
=974 \$aPython Collection$9LOCAL

=LDR 00115 a2200061 4500
=001 59231572
=949 \$a999999999998
=974 \$aMath Collection$9LOCAL
Why a function and command line argument?

Enabling the input file to be specified on a command line allows you to run the same script over different files without requiring script editing skills.

For example, you and others in your department might have a monthly process involving cleaning up vendor metadata from several different vendors. The python script you’re using is installed on a shared drive or server.

By enabling the input file to be specified on the command line, you and your colleagues can place the input files you each are using and process them via the command line without opening up / editing the script.
Tutorials and Further Reading

https://www.learnpython.org/

https://developers.google.com/edu/python/


Questions?

Twitter: @lpmagnuson

lmagnuson@csusm.edu