Wrangle Your Data Like a Pro With the Data Processing Power of Python

Geoffrey P. Timms
College of Charleston, timmsgp@gmail.com

Jeremy M. Brown
Mercer University Libraries

Wrangle Your Data Like a Pro With the Data Processing Power of Python

Geoffrey P. Timms, Librarian for Marine Resources, College of Charleston/SCDNR/NOAA

Jeremy M. Brown, Assistant Dean for Technical Services and Systems, Mercer University

Abstract

Management, delivery, and marketing of library resources and collections necessitate interaction with a plethora of data from many sources and in many forms. Accessing and transforming data into meaningful information or different formats used in library automation can be time consuming, but a working knowledge of a programming language can improve efficiency in many facets of librarianship. From processing lists to creating extensible markup language (XML), from editing machine-readable cataloging (MARC) records before upload to automating statistical reports, the Python programming language and third-party application programming interfaces (APIs) can be used to accomplish both behind-the-scenes tasks and end-user facing projects. Creating programmatic solutions to problems requires an understanding of potential. Here we summarize the data sources, flows, and transformations used to accomplish existing projects at Mercer University and the College of Charleston. Foundational programming techniques are explained and resources for learning Python are shared.

Introduction

Libraries generate and have access to more data than ever before, which presents exciting opportunities to take library services to new heights and enhance or personalize patrons' experiences. All it takes is data transformation. There are many reasons why a librarian might seek to transform data: Perhaps to add value by presenting them in a different form, or to prepare them structurally for ingestion into a database or library automation system. Unfortunately, working with data can be a time-consuming process. Some tasks, such as documenting library use, may be demanded monthly. Others, such as processing e-resource usage data, may only be tackled annually but consist of great quantities of data. Whatever the reason, data transformation is undertaken to restructure, synthesize, or substitute data to achieve a higher level of utility.

The transformation of data is part of a process that can be automated if data is available in a machine-readable form and is of a predictable structure. Ideally, data should be accessible with an automated process, and the transformed data should be deliverable in an appropriate format and location automatically. However, this is not always technically possible, and human intervention may be required during the transformation process. Regardless, automated transformation can accomplish in fractions of seconds what may consume hours of a person's time.

Once one or more datasets are obtained, they must be ingested by a script and then organized in a manner suitable for the transformation process. The transformation itself may be as simple as rearranging the data or tidying them by removing superfluous data. Transformation may involve generating visual representations of data or using parts of the data to reference other information stored online, such as book cover art. Once transformed, the output is then delivered in the appropriate form, be it XML, HTML, or images for on-screen display; stored as data in an SQL database; or inserted into a file. The following examples of data transformation projects demonstrate the data sources and the output of new information. See Appendix for figures, data, and tables.

Virtual New Bookshelf

- Transformation summary: Bibliographic data transformed to XML for Web display.
- Data input: Text file manually downloaded from library catalog (Table 1) consisting of record number, ISBN string, title, call number, and publication information for recently purchased books.

https://doi.org/10.5703/1288284316489

- Output: Write XML to an XML file (Figure 1), which is then styled using XSL for HTML display (Figure 2).

Map of the Library Shelving Highlighting the Range for a Chosen Item

- Transformation summary: Call number and location code used to create a visual map of the chosen item's location.
- Data input:
    - Catalog record for chosen item information (Figure 3).
    - SQL database tables including:
        - Range number, location code, call number of first book on range (Table 2).
        - Map coordinates of each range in each location code (Table 3).
        - Waypoints for each range in each location code (Table 4).
    - Base map image file for each floor showing all ranges.
- Output: Return custom map image to the screen (Figure 4).

Conversion of Monthly Transaction Logs to a Multisheet Excel Summary Report

- Transformation summary: Transaction logs from ILL borrowing and lending, and reference analytics transformed to multisheet Excel workbook documenting multiple agencies' use of the library.
- Data input: CSV and Excel files manually downloaded monthly from Interlibrary Loan (ILLiad) (Table 5) and Reference Analytics.
- Output: Excel workbook with three summary report sheets (Table 6).

Foundational Programming Techniques

Some programming techniques are fundamental to most of the projects that the authors have undertaken. Several are presented below.

Variables, Dictionaries, and Lists

In Python, as in most programming languages, we deal with data of various types: Among other types, there are integers (whole numbers), floating point numbers (such as 3.14), and strings (alphanumeric data of any length). Python detects the type of variable based upon the data that we put into it. For example:

```
integer_value = 4
string_value = 'dog'
float_value = 3.14
```

Frequently, we deal with data that makes sense to group together. For example, we need to make a list of similar values. This list could represent a row or a column of a spreadsheet, or it could simply be a list of values. In Python, we create a list with square brackets, separating the values by commas:

```
a_list = [1, 3, 4, 5, ]
```

Proper form dictates that we leave a trailing comma, in case we want to add another value. We could create an empty list in a couple of different ways:

```
a_list = []
another_list = list()
```

Lists are mutable, meaning they may be changed after creation, so we could take our list, and add an element to it:

```
a_list.append(6)
```

Or we could sort it:

```
a_list.sort() # sorts in ascending order
a_list.sort(reverse=True) # sorts in descending order
```

To access data in our list, we simply call it by its position within the list:

```
a_list[0]
```

In our first list, which contained the values 1 through 5, we would receive the first one, which is the number 1. Note that list indexes always begin with zero.

Other times, we have data that belongs together but needs some amount of compartmentalization. In

Python, a useful data structure for that kind of data is a dictionary. In Python, a dictionary is a pair of data that is made up of a key and a value. Typically, a simple data type (such as a string or an integer) is used as the key, but the value could be anything. Curly braces denote the beginning and end of a dictionary. For example:

```
a_dictionary = {
  'name' : 'Hans Mustermann',
  'surname' : 'Mustermann',
  'given_name' : 'Hans',
  'age' : 35,
  'children' : ['Frank','Annie','Jim Bob',]
          }
```

In the above example, note that we have a mix of data types. We have three strings: name, surname, and given_name, followed by an integer (age) and then a list (children). Values that we store in dictionaries can be retrieved with the same syntax as the list uses:

```
a_dictionary['children']
```

This would give us the list we stored:

```
['Frank','Annie','Jim Bob',]
```

The thing to remember about dictionaries is that there is no guarantee that anything will be stored in any particular order.

Iterating Over/Comparing Variables and Lists

Iterating over a list allows us to examine each element in the list and potentially do something with it. Consider a case where we declare a list full of numbers and want to print each number on its own line:

```
a_list = [1,2,3,4,5]

for item in a_list:
  print(item)
```

This "for" statement proceeds through a list and assigns a value to "item." This is convenient when we just need to use the item and move on. If, however, we want to compare two lists of values to see how many times items in one list appear in another list, we must proceed differently. We declare two lists: a data list and a list with values to look for and count. Since the second list really has two items for each element (a value and a count), we could use a dictionary to keep these values separate, so we put together a list of dictionaries.

```
data_list = [6, 5, 9, 1, 8, 2, 9, 7,
        1, 4, 4, 2, 1, 9, 3, 1,
        3, 5, 4, 7, 7, 2, 5, 1,
        6, 7, 3, 2, 4, 9, 8, 3,
        2, 5, 6, 6, 9, 7, 1, 8,
        6, 1, 2, 8, 7]
a_list = [
      {'value':1, 'count':0},
      {'value':2, 'count':0},
      {'value':3, 'count':0},
      {'value':4, 'count':0},
      {'value':5, 'count':0}
      ]
```

Now that we have our data, we can prepare to look through our two lists. Because we have two lists, we will need to have two loops. This is called a nested loop because one loop is within the other loop:

```
for item in a_list:
  for r_item in data_list:
    if item['value'] == r_item:
      item['count'] += 1
    else:
      pass # do nothing
  print(item)
```

The first loop assigns each value of a_list to the variable "item." Then, we immediately proceed into the data_list. We assign a value from data_list to the variable "r_item," and then we compare the value of the item to r_item. We use the "equals" operator, which is a double equal sign (==). If the two integer values are equal, this returns true, and we then add one to item's count argument. If the two items are not equal to each other, we don't do anything. Instead, we just move on to the next element in data_list. When we have moved through all 45 values in data_list, we print the values that we just stored in item, and then move on to the next item in a_list and repeat the comparison process. This renders the output:

```
{'count': 7, 'value': 1}
{'count': 6, 'value': 2}
{'count': 4, 'value': 3}
{'count': 4, 'value': 4}
{'count': 4, 'value': 5}
```

Reading and Writing Files (Input/Output)

In practice, we encounter data files that contain lists. These could originate from Excel, or they could be reports from another system. Frequently, the data fields are separated by special characters, such as commas, tabs, or some other character. In this example, we are reading two lists of ISSN data. The task consists of the following steps:

1. Read in all the lines of two files.
2. Create results.
3. Output this to a new file

```
with open('input1.csv','rU') as L1:
   reader = csv.reader(L1, delimiter="\t")
   list1 = []
   for sublist in list(reader):
      list1.append(sublist.pop())
      list1.pop(0)  #Remove the first row from
list1, which is the column names
```

The code above gets us a file handle (L1) pointing at the data file. We point a CSV reader at that, configured to handle tab-delimited data. We create an empty list (list1) to store our data. The for loop is dense code: We are converting all lines in the file to a list. Each list here represents the rows in a spreadsheet, and each contains a sublist of column values. Because we only have one column of data, we can pop the only item off the sublist and append it to our results list, list1. This takes care of our input.

Once we have processed our input, we will need to output the data. For this example, we have our output in a dictionary called finaldata, which we wish to store in a Microsoft Excel file. Using xlwt makes this almost as simple as the prior example:

First, we create a workbook, and name a sheet in it. Then, we write our two column headings. Finally, we loop over our data set, writing our ISSN value in column 0 and our title in column 1. Then, we save the workbook.

```
wb = xlwt.Workbook()
Sheet1 = wb.add_sheet('Journals found in both
lists')
Sheet1.write(0,0, 'Linking ISSN-L')
Sheet1.write(0,1, 'Title')
n=1
for issn,title in finaldata.items():
   Sheet1.write(n,0,issn)
   Sheet1.write(n,1,title)
   n+=1
```

Regular Expressions (Pattern Matching)

In our case study, we face data of varying formats. Really, the values are all the same, but punctuation and notation differ. As long as this is somewhat regular, we can anticipate this variance and obtain the data we need. Pattern matching is a very broad topic in itself, but it is simple to learn the basics and begin extracting data.

Before we begin, it is important to realize that most every character we can type will match itself and become part of the pattern. A “2” generally matches a typed number two, for example. However, we need to be aware of some special characters that are used in the syntax of this powerful programming language: .^$*+?-{}[]()\|. These each have special meanings.

Before we can start using regular expressions in Python, we need to import the library using an import statement:

```
import re
```

We can then proceed to use it in our subsequent code. We will focus on the convenience functions, re.sub() and re.match(). Consider the case of an ISSN. An ISSN is typically eight meaningful digits, except when the final character also happens to be an X. It could be written in several ways. Let us consider three cases:

1. Written with a hyphen between two sets of four characters: 0022-510X.
2. Written without the hyphen: 0022510X.
3. There could be extra characters before or after our ISSN: ISSN: 0022-510X *Journal of the Neurological Sciences*.

An easy way to attack this problem is to detect which case we have and deal with that with an if/elif statement. However, we have to use the match function first. Match returns a match object if it does indeed match something, or if it matches nothing, it returns a value of None. If we receive a value of None, we know we did not get a match.

Examining the first pattern, we see four digits in a row followed by a hyphen, followed by three digits and an X. That X could also be a digit, but this informs us as to how to write our pattern:

```
re.match(r'(\d{4})-(\d{3}[0-9Xx])','0022-510X')
```

The string itself specifies that we want two groups of characters: Four digits, a hyphen, and three digits followed by anything zero through nine, or an upper or lowercase X. The second value is our ISSN string.

In our second case, perhaps we have data where we do not know if we have a hyphen or not. Both could be correct, but if we want to match that, we could use much the same search string, but then address the number of hyphens. We could have zero hyphens, or we could have a single hyphen. This statement detects either of those cases:

```
re.match(r'.*(\d{4})\-{0,1}(\d{3}[0-9Xx])', '0022510X')
```

Our third and final case is an ISSN with extra text before or after the ISSN. We can craft a search pattern that looks for any character (a period: .) any number of times (an asterisk: *), and include that before and after our search pattern:

```
r'.*(\d{4})\-{0,1}(\d{3}[0-9Xx]).*'
```

The next step is to clean up that regular expression. We can use the re.sub() function to do just that. *Sub* stands for substitution, and it allows us to use regular expressions to replace characters within a string. There are three arguments that we need to provide to this function:

1. A pattern, which we have already seen in our handling of the match function.
2. A substitution pattern, and finally
3. A string to examine.

The substitution pattern is really the only new thing here. In our regular expression that perfectly matches our three cases, we already have two sets of parentheses. Each set captures a portion of the ISSN, and each portion is exactly four characters. We could add that to a substitution call and cause it to reformat any of the three cases into our preferred format: 0022-510X, which looks like the following:

```
re.sub(r'.*(\d{4})\-{0,1}(\d{3}[0-9Xx])',r'\1-\2','ISSN: 0022-510X Journal of the Neurological Sciences')
```

Using Data from APIs

The acronym API is used very frequently in software circles. These days it seems that it has penetrated our conversations a little further, since many publishers and most systems vendors tell us they have their own APIs that we only need to reach out and use. API stands for application programming interface, and defined, an API provides a means for a programmer to write a program that uses their software.

Since an API is essentially a way for us to communicate with another application, we need to know what data to send it and what to expect from the API when it gives us data back. In our next case study, we use the OCLC xISSN Web service (http://xissn.worldcat.org/xissnadmin/doc/api.htm). The documentation tells us that we can obtain journal metadata from the API (namely, we want the title that corresponds to the ISSN) if we query it using a URL of the form http://xissn.worldcat.org/webservices/xid/issn/0022-510X?method=getEditions&format=json

There are many ways to access URLs in Python. A popular one is to use the Requests library (http://docs.python-requests.org).

```
import requests, json

url = 'http://xissn.worldcat.org/webservices/xid/issn/0022-510X?method=getEditions&format=json'
r = requests.get(url)
formatted_json = json.dumps(r.json(), sort_keys=True, indent=4, separators=(',', ': '))
print(formatted_json)
```

In this example, we first import the two libraries we need: requests and JavaScript Object Notation (JSON). Then we define a URL. Next, we create a requests object for our URL. This requests object fetches the data from the URL when we create it, so this is where we have done all the heavy lifting. At this point, we have all the data we need, but long blocks of JSON are somewhat tough to read. The next thing is to reformat the JSON so we can easily read it. We use Python's built-in JSON library to sort the keys, indent each level four characters, and use a comma as a list separator and the colon as the assignment indicator.

When we examine this output (Appendix, Figure 5) we see two data structures: stat and group. Inside "group," we find our data. The trick to accessing this data is to pry the dictionary apart. If we want the title, we would have to access it from within the dictionary contained in list. This looks like so:

`r.json()['group'][0]['list'][0]['title']`

This is because the title is contained by a dictionary and contained by a list, which is in a dictionary called "list," which is a key in a dictionary contained within a list called "group," which is a key in a dictionary.

Case Study

The following case study utilizes the above programming techniques. Initial data are provided in two differently formatted lists of ISSNs stored in two CSV files (Appendix, Table 7). One list (~11,000 ISSNs) represents a broad array of journals. The other list (49 ISSNs) represents locally held titles. The lists are transformed into an Excel spreadsheet that presents the corresponding linking ISSNs (ISSN-Ls) and journal titles only for the ISSNs that occurred in both lists, using an OCLC API. The Python code is broken into discreet chunks called functions, representing each phase of the transformation. The full code can be found at http://libraries.mercer.edu/ursa/handle/10898/3687.

Phase One

The make_lists function opens each CSV file in turn, reads it as a tab-delimited structure, and puts its content into a list.

Phase Two

The clean_list function is called once for each list of ISSNs. Each list item is taken in turn and tested against a regular expression to see if a properly formed ISSN can be extracted. If so, it is reformatted and added to a new list. Otherwise, an error is reported, and the script moves to the next item.

Phase Three

The get_dupelist function compares both cleaned lists to find duplicate ISSNs using an internal method called "intersection." To use this method, each list must be converted into sets. Sets differ from lists because they are unordered and can contain no duplicates. The intersection method results in a set of items found in both sets. This set is converted to a list for use in Phase Four.

Phase Four

The get_oclc function is used to create a dictionary of key:value pairs representing ISSN-L (the key) and journal title (the value) obtained from the OCLC API described earlier. For each ISSN in the list generated in Phase Three, the API URL is formed using the ISSN, and the request is sent. The ISSN-L and title are extracted from the response.

Phase Five

The write_sheet function creates an Excel workbook and sheet with column headings. It then iterates over the data dictionary to write the key:value pairs of ISSN-L and title to the spreadsheet. Once every dictionary item has been written, the spreadsheet is saved, and the operation is complete (Appendix, Table 8).

Communicating With Systems People

Realistically, most complex programming will be undertaken by those who already possess the experience to deliver programmatic solutions to particular problems. As such, a data transformation project will likely involve communication between detail-oriented programmers and other staff who may have a different perspective. In order to help a programmer deliver a solution to a problem, it is important to be ready to:

- Explain what you want to accomplish.
- Describe the desired end result in detail.
- Detail data sources, formats, and locations.
- Outline data transformations.
- Discuss potential data anomalies.
- Provide data samples.

Participating in such a project provides the perfect opportunity to contribute to and learn about the solution put into place by the programmer. Such inquiry may also contribute to one's own development as a novice programmer. Learning by doing is perhaps the best way to get started.

Getting Started With Python

Python may be installed and operated on a workstation or server. Python libraries may also be added in both these contexts. For Python to interact with SQL-based databases, software such as PostgreSQL, MySQL, or SQLite must also be installed. User interaction with Python may be undertaken in a variety of ways. An integrated development environment (IDE) is separate software used to make the creation, testing, and preservation of code as convenient as possible. Alternatively, Python may be developed and run from a command line interface, which is a less convenient method.

Software

- Python: https://www.python.org/
- Python's built-in libraries: https://docs.python.org/3/library/
- Eclipse IDE: https://eclipse.org/ide/
- SQL databases
    - MySQL: https://www.mysql.com/
    - SQLite: https://sqlite.org/
    - PostgreSQL: https://www.postgresql.org/

Resources for Learning Python

- A Byte of Python: https://python.swaroopch.com/
- Codecademy: https://www.codecademy.com/learn/python
- Introduction to Programming with Python: http://opentechschool.github.io/python-beginners/en/index.html
- Python Practice Book: http://anandology.com/python-practice-book/
- Stack Overflow discussion list: http://stackoverflow.com/questions/tagged/python

Conclusion

Data transformation with Python may be achieved at various levels of complexity. At its simplest, we have cleaned and compared two lists of data and referenced an API. Our examples of more complex projects undertaken at our libraries recently demonstrate the potential for transformation of data into visual or graphical information. Successfully learning a programming language is within reach of self-motivated individuals who possess a curious and systematic mindset. Libraries that are fortunate enough to have programmers on staff will be better able to utilize these unique skills when nonprogrammer staff have a basic appreciation for the potential offered by programmatic solutions to library tasks and problems involving data.

Appendix

Table 1. Bibliographic data exported from Library Management System.

RECORD #(BIBLIO)	STANDARD #	TITLE	CALL #(BIBLIO)	PUB INFO
b26619878	1489976205, "9781489976208", "9781489976215 (e-Book)"	Protein NMR: Modern techniques and biomedical applications / Lawrence Berliner editor.	QP551 .P7684 2015	New York: Springer [2015].
b26623754	9781493922901 (alk. paper)," 1493922904 (alk. paper)", "9781493922918 (e-Book)", "1493922912 (e-Book)"	RNA bioinformatics / edited by Ernesto Picardi.	QH324.2 .R53 2015	New York: Humana Press [2015].
b2662381x	9780128025086, "0128025085"	Emerging trends in computational biology bioinformatics and systems biology: Algorithms and software tools / edited by Quoc-Nam Tran & Hamid Arabnia.	QH324.2 .E455 2015	Waltham, MA: Elsevier: Morgan Kaufman [2015].
b26625969	9781441979759 (pbk.), "1441979751 (pbk.) "	Numerical ecology with R / Daniel Borcard, Francois Gillet, & Pierre Legendre.	QH541.15.S72 B67 2011	New York: Springer [2011].
b26652675	9781493925490, "1493925490", "9781493925506 (ebk.)","1493925504 (ebk.)"	Proteomic profiling: Methods and protocols / edited by Anton Posch.	QP551 .P7566 2015	New York: Humana Press [2015].
b26652687	0444633987,"9780444633989"	Estuarine ecohydrology: An introduction / Eric Wolanski & Michael Elliott.	QH541.15.E19 W65 2016	Amsterdam Netherlands: Elsevier [2016].

```
<?xml version='1.0' encoding='UTF-8'?>
<xml><head_title>New Books for the Last Quarter</head_title><updated>updated November 02,
2016</updated>
<items>
        <item>
                <link name="link">http://libcat.cofc.edu/record=b2661987~S7</link>
                <isbn name="isbn">1489976205</isbn>
                <title name="title">Protein NMR : modern techniques and biomedical applications / Lawrence
        Berliner  editor.</title>
                <callno name="callno">QP551 .P7684 2015</callno>
                <pub name="pub">New York : Springer  2015.</pub>
                <code name="code">1</code>
        </item>
        <item>
                <link name="link">http://libcat.cofc.edu/record=b2662375~S7</link>
                <isbn name="isbn">1493922904</isbn>
                <title name="title">RNA bioinformatics / edited by Ernesto Picardi.</title>
                <callno name="callno">QH324.2 .R53 2015</callno>
                <pub name="pub">New York : Humana Press  [2015]</pub>
                <code name="code">2</code>
        </item>
        <item>
                <link name="link">http://libcat.cofc.edu/record=b2662381~S7</link>
                <isbn name="isbn">0128025085</isbn>
                <title name="title">Emerging trends in computational biology  bioinformatics  and systems
        biology : algorithms and software tools / edited by Quoc Nam Tran  Hamid Arabnia.</title>
                <callno name="callno">QH324.2 .E455 2015</callno>
                <pub name="pub">Waltham  MA : Elsevier : Morgan Kaufman  [2015]</pub>
                <code name="code">2</code>
        </item>
```

Figure 1. XML output following transformation.

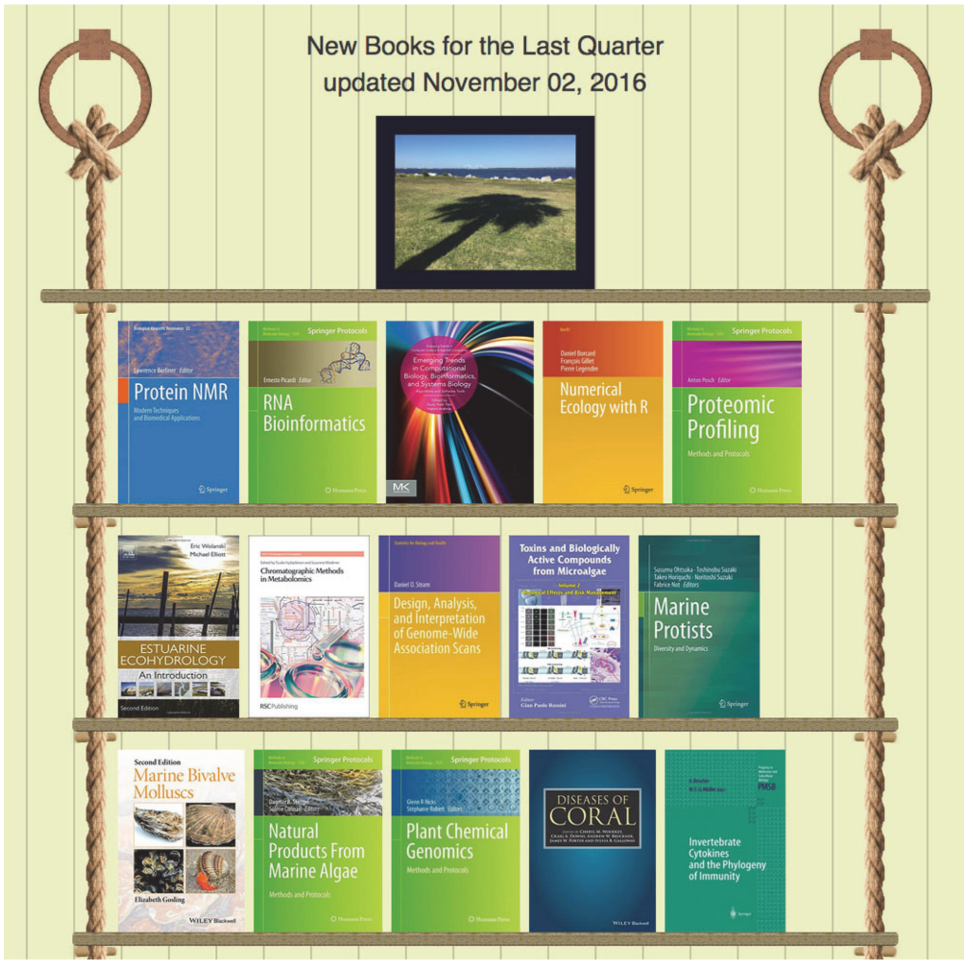
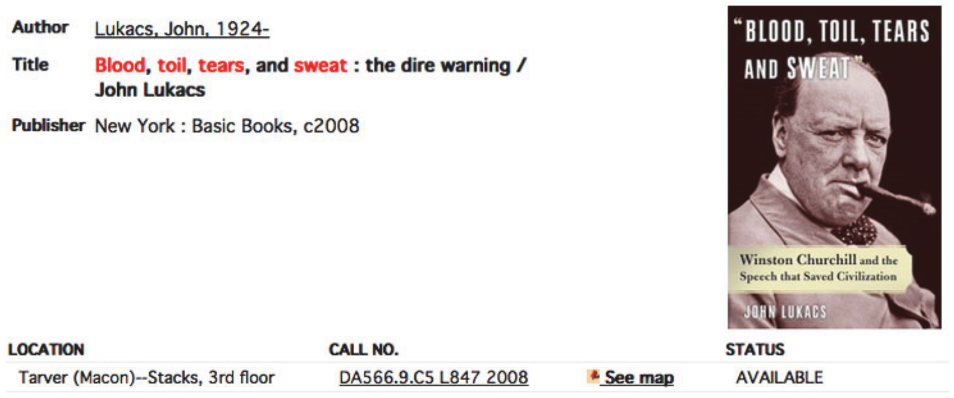

New Books for the Last Quarter
updated November 02, 2016
Protein NMR
RNA Bioinformatics
Emerging Trends in Computational Biology, Bioinformatics, and Systems Biology
Numerical Ecology with R
Proteomic Profiling
Estuarine Ecohydrology
Chromatographic Methods in Metabolomics
Design, Analysis, and Interpretation of Genome-Wide Association Scans
Toxins and Biologically Active Compounds from Microalgae
Marine Protists
Marine Bivalve Molluscs
Natural Products From Marine Algae
Plant Chemical Genomics
Diseases of Coral
Invertebrate Cytokines and the Phylogeny of Immunity

Figure 2. Virtual bookshelf web page resulting from XML/XSL data.



Author	Lukacs, John, 1924-
Title	**Blood, toil, tears, and sweat : the dire warning / John Lukacs**
Publisher	New York : Basic Books, c2008

LOCATION	CALL NO.		STATUS
Tarver (Macon)--Stacks, 3rd floor	DA566.9.C5 L847 2008	See map	AVAILABLE

"Blood, Toil, Tears and Sweat"
Winston Churchill and the Speech that Saved Civilization
John Lukacs

Figure 3. Catalog record excerpt for a book, showing location and call number.

Table 2. PostgreSQL data excerpt for shelf ranges showing collection and normalized call number for the first item on each shelf range.

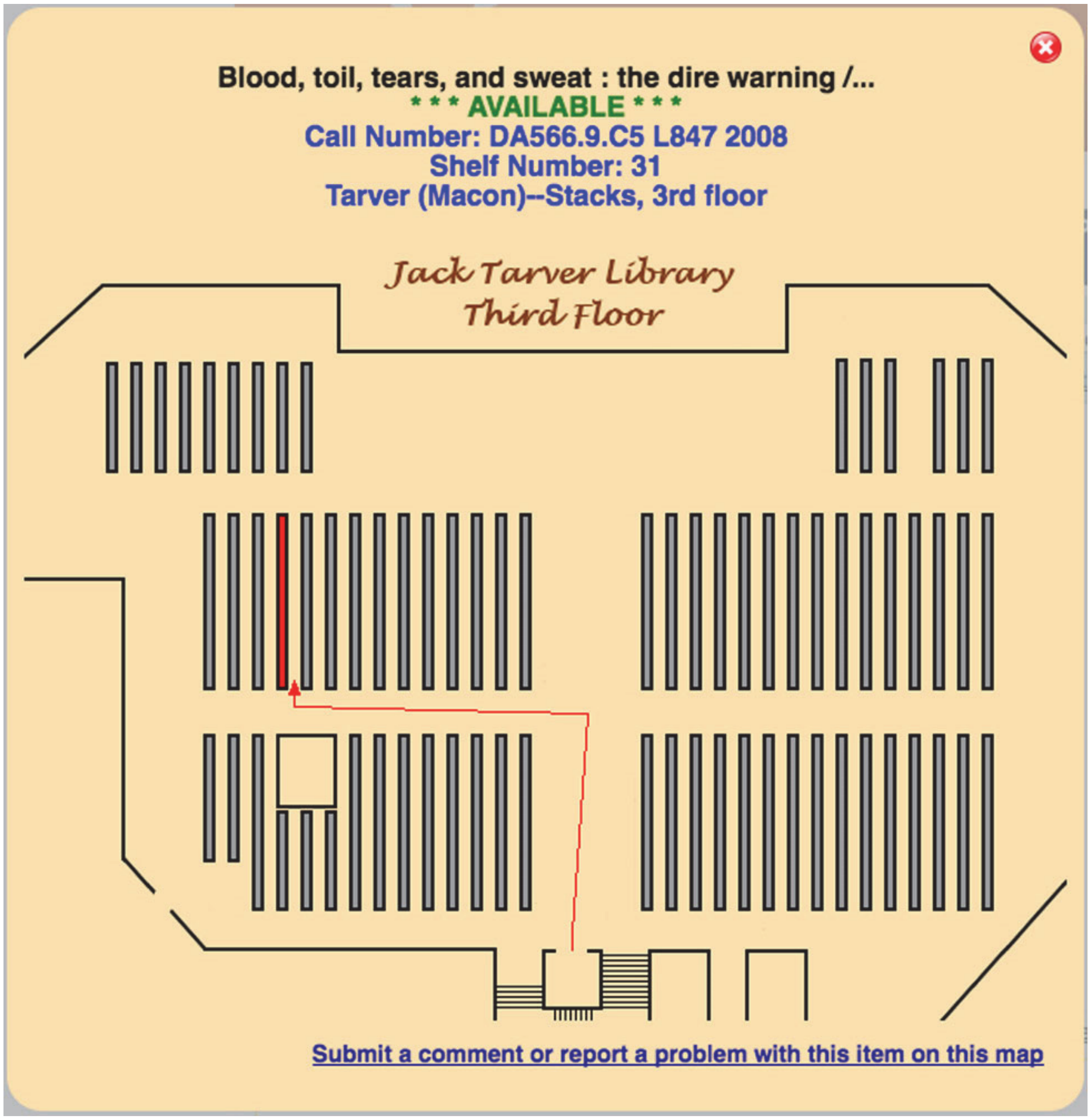
id	shelf	location	call_start
24	24	T3mstk	BX0877400S96000
25	25	T3mstk	D00000200A70000
26	26	T3mstk	D00052100H35000
27	27	T3mstk	D00075700A64000
28	28	T3mstk	D00080430L35700
29	29	T3mstk	DA0022800L30000
30	30	T3mstk	DA0039600A20000Y60000
31	31	T3mstk	DA0053000N53000
32	32	T3mstk	DC0010300S20000

Table 3. PostgreSQL data excerpt for shelf ranges showing collection location, map coordinates, arrowhead aspect (aisle), waypoints, and fill style.

id	shelf	location	coordinates	aisle	waypoints	style
24	23	T3mstk	[[133,287,135,287,135,384,133,384]]	w	[300,301]	f
25	24	T3mstk	[[133,287,135,287,135,384,133,384]]	e	[300,301]	f
26	27	T3mstk	[[147,73,149,73,149,132,147,132]]	w	[300,301,302,303]	f
27	32	T3mstk	[[147,73,149,73,149,132,147,132]]	e	[300,301,302,303]	f
28	28	T3mstk	[[147,160,149,160,149,257,147,257]]	w	[300,301,302]	f
29	31	T3mstk	[[147,160,149,160,149,257,147,257]]	e	[300,301,302]	f
30	29	T3mstk	[[147,331,149,331,149,384,147,384]]	w	[300,301]	f
31	30	T3mstk	[[147,331,149,331,149,384,147,384]]	e	[300,301]	f
32	33	T3mstk	[[161,73,163,73,163,132,161,132]]	w	[300,301,302,303]	f

Table 4. PostgreSQL data excerpt for waypoints showing waypoint coordinates.

id	wpid	waypointcoords
5	104	[302,226]
6	105	[302,210]
7	106	[302,83]
8	107	[415,83]
9	200	[315,409]
10	201	[315,400]
11	202	[315,340]
12	203	[315,315]
13	204	[315,245]



Blood, toil, tears, and sweat : the dire warning /...

* * * AVAILABLE * * *

Call Number: DA566.9.C5 L847 2008

Shelf Number: 31

Tarver (Macon)--Stacks, 3rd floor

Jack Tarver Library
Third Floor
Submit a comment or report a problem with this item on this map

Figure 4. Map of library third floor showing shelf range location following data synthesis and transformation.

Table 5. Edited excerpt of Interlibrary Loan borrowing transaction log.

Request Type	Transaction Status	Transaction Date	Status	Department
Article	Request Finished	10/14/16	Staff	NOAA Hollings Marine Laboratory
Article	Request Finished	10/13/16	Faculty	Library
Article	Delivered to Web	10/25/16	Faculty	GPMB/Grice Marine Laboratory
Article	Request Finished	10/29/16	Faculty	NOAA CCEHBR Charleston
Article	Request Finished	10/8/16	Faculty	GPMB/Grice Marine Laboratory
Article	Request Finished	10/29/16	Faculty	Library
Article	Request Finished	10/24/16	Faculty	NOAA CCEHBR Charleston
Loan	Cancelled by ILL Staff	10/12/16	Faculty	Library
Article	Request Finished	10/24/16	Faculty	NOAA CCEHBR Charleston
Article	Request Finished	10/26/16	Staff	NOAA CCEHBR Oxford
Article	Delivered to Web	10/27/16	Faculty	GPMB/Grice Marine Laboratory
Article	Request Finished	10/24/16	Faculty	NOAA CCEHBR Charleston

Table 6. Interlibrary Loan borrowing summary report following data transformation.

		Agency	Oct
Requested			
	Book		
		CofC	1
		SCDNR	0
		NOAA	0
		NOAA OCM	0
		NIST	0
	Article		
		CofC	10
		SCDNR	0
		NOAA	12
		NOAA OCM	0
		NIST	0
Received			
	Book		
		CofC	0
		SCDNR	0
		NOAA	0
		NOAA OCM	0
		NIST	0
	Article		
		CofC	7
		SCDNR	0
		NOAA	10
		NOAA OCM	0
		NIST	0

```
{
  "group": [
    {
      "list": [
        {
          "form": "JD", # Digital Journal
          "issn": "1878-5883",
          "issnl": "0022-510X", # Linking ISSN
          "oclcnum": [
            "473214202",
            "67256818",
            "39191656",
            "876153175"
          ],
          "peerreview": "Y",
          "publisher": "Amsterdam : Elsevier Science",
          "rawcoverage": "Began with v. 1, issue 1 (Jan./Feb. 1964).",
          "rssurl": "http://rss.sciencedirect.com/publication/science/4854",
          "title": "Journal of the neurological sciences"
        },
      ],
      "rel": "this"
    }
  ],
  "stat": "ok"
}
```

Figure 5. Output from OCLC xISSN API.

Table 7. Sample of raw ISSN data from each of two CSV files.

ISSNs
02652048
02506807
16101928
13684302
07415214
1721727X
05874254
00014575
17588251
00218456

Local Holding ISSNs
ISSN: 1387-974X
ISSN: 1110-662X
ISSN: 0022-3131
ISSN: 1103-8128
ISSN: 0257-9731
ISSN: 0368-492X
ISSN: 1383-469X
ISSN: 00297828
ISSN: 0015-7120
ISSN: 0742-4787

Table 8. Sample of the final ISSN-L and title data in Excel.

Linking ISSN-L	Title
1523-3790	Current oncology reports
1096-6218	Journal of palliative medicine
0014-4797	Experimental agriculture
0102-8650	Acta cirúrgica brasileira / Sociedade Brasileira para Desenvolvimento Pesquisa em Cirurgia
1103-8128	Scandinavian journal of occupational therapy
1387-974X	Photonic network communications
1610-8167	Urban forestry & urban greening
0737-3937	Drying technology
1368-4221	The econometrics journal
1742-4755	Reproductive health RH