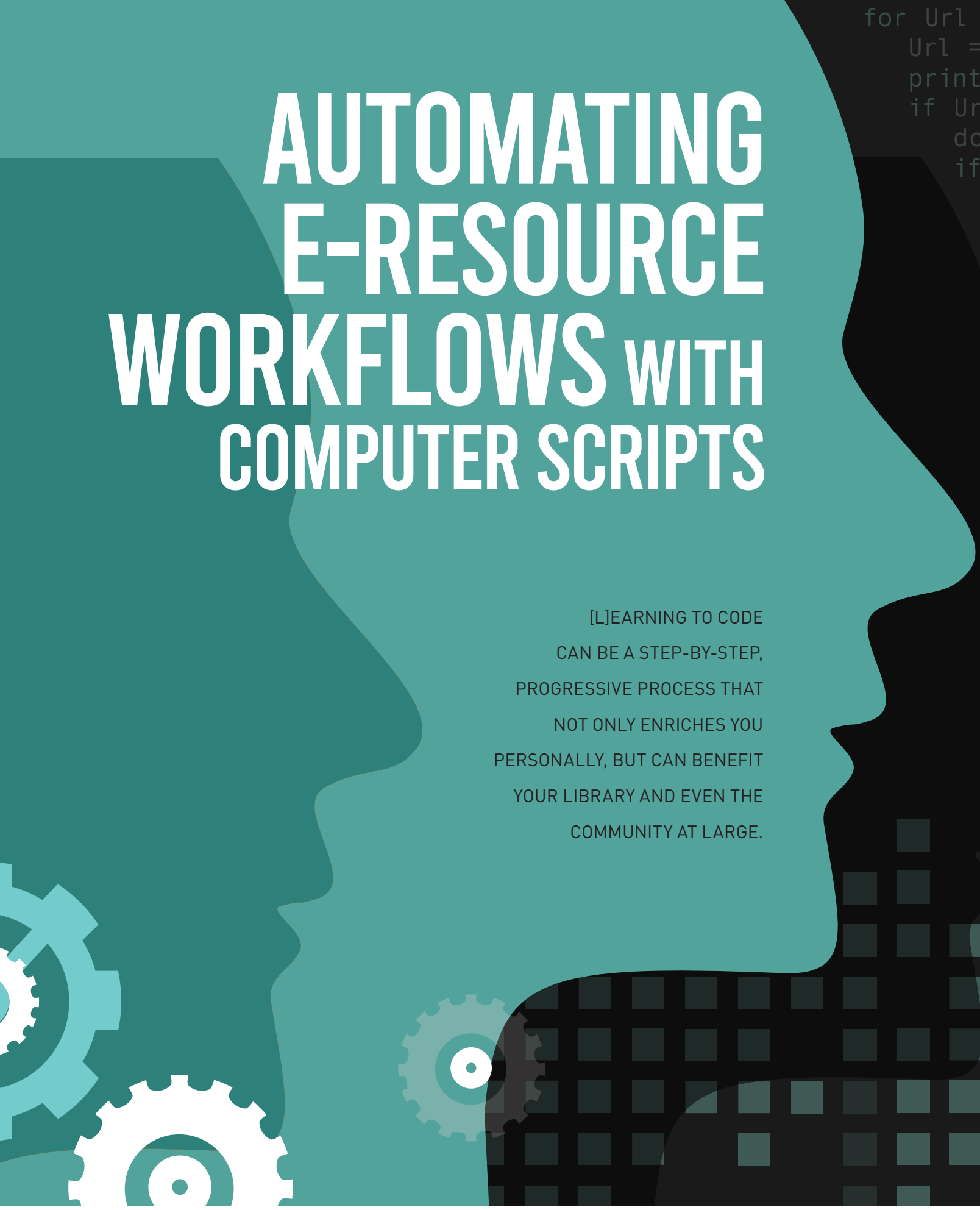
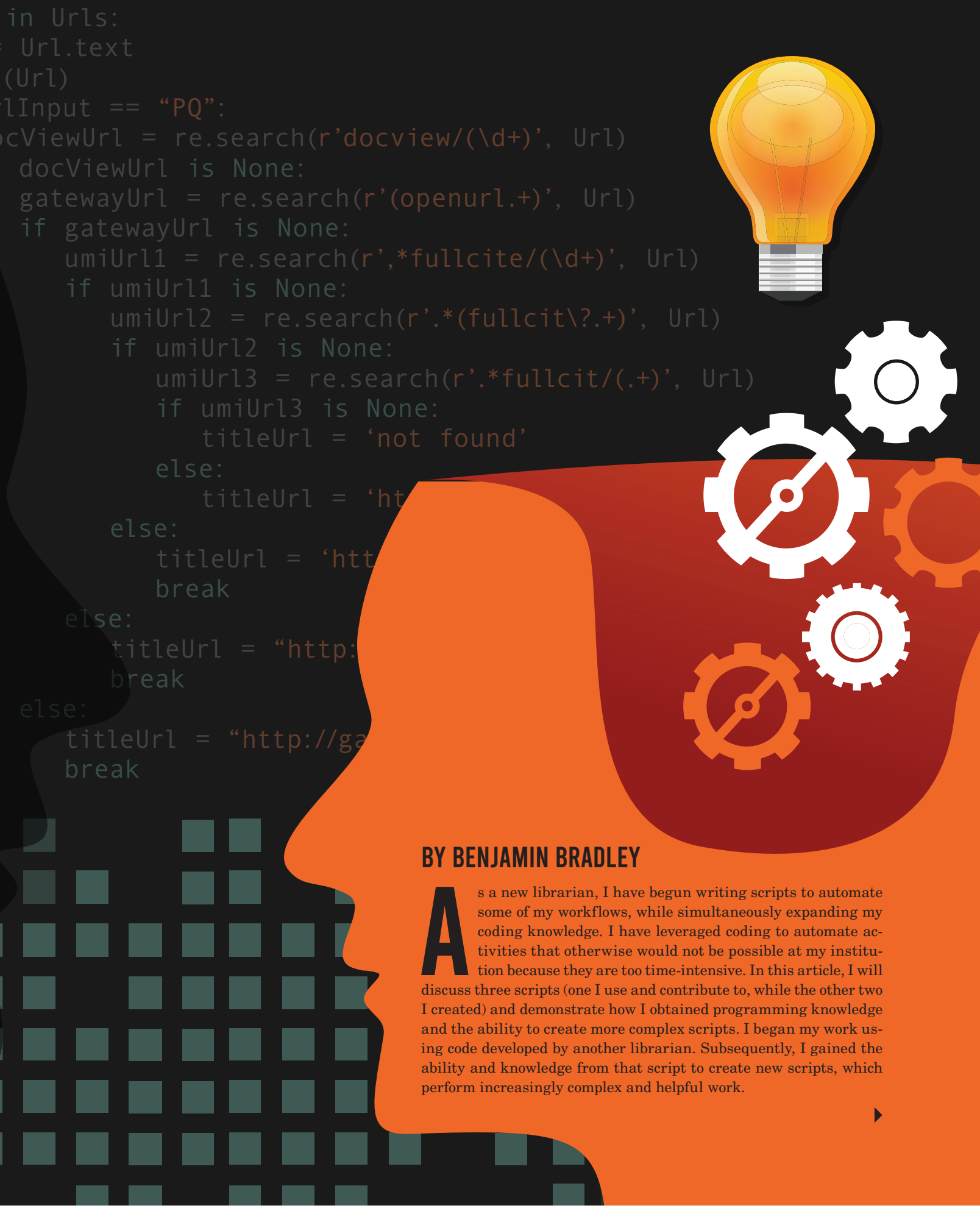
AUTOMATING E-RESOURCE WORKFLOWS WITH COMPUTER SCRIPTS



[L]EARNING TO CODE CAN BE A STEP-BY-STEP, PROGRESSIVE PROCESS THAT NOT ONLY ENRICHES YOU PERSONALLY, BUT CAN BENEFIT YOUR LIBRARY AND EVEN THE COMMUNITY AT LARGE.



BY BENJAMIN BRADLEY

As a new librarian, I have begun writing scripts to automate some of my workflows, while simultaneously expanding my coding knowledge. I have leveraged coding to automate activities that otherwise would not be possible at my institution because they are too time-intensive. In this article, I will discuss three scripts (one I use and contribute to, while the other two I created) and demonstrate how I obtained programming knowledge and the ability to create more complex scripts. I began my work using code developed by another librarian. Subsequently, I gained the ability and knowledge from that script to create new scripts, which perform increasingly complex and helpful work.

Ebook Access Checker

When I started in my current position as a discovery librarian at the University of Maryland (UMD) Libraries, my supervisor introduced me to the ebook Access Checker[1] and asked me to use it to evaluate some ebook collections. Developed by Kristina Spurgin[2] and written in JRuby, the Access Checker enables librarians to perform automated link checking and assesses if the vendor is providing access to the title. The script outputs a report listing if the link worked and the library has access or if there was a problem. The librarian can fix the problem or follow up with the appropriate vendor. Librarians usually need their users to report these kinds of problems; by utilizing the script, librarians are able to transform this reactive maintenance activity into a proactive process to better serve their users. The script reads links from a text file, which can be a simple list of URLs, or you can use a title list or KBART file. The only requirement is that the URLs must be in the last column of the file, so some minor editing may be required. Once the script is running, it asks users to select which platform they are using, and then it will begin to check the links (see Figure 1).

When selecting a platform, the user is triggering different sections of code that apply certain criteria to evaluate access. See Figure 2 for an example of code for Alexander Street Press.

The logic for packages act as templates, allowing someone to learn how the script works and add new platforms. The logic for packages use two methods: "include" and "match." The include method takes a string, while match uses a regular expression pattern to match against the platform's code and then tries to find it in the webpage's code. If a match is found, the access variable takes the appropriate value. By understanding these two methods as used in the existing code, I could create code for other packages. See Figure 3 for code I added for the IGI Global platform.

To construct the logic for the platform, I looked at the IGI platform to find code to use in the script. It displays a green check mark on book chapters when access is available to the user. Using Element Inspect in my browser, I found the code associated with the green check marks, which I used for the script (see Figure 4).

```
What platform/package are you access checking?
Type one of the following:
  asp    : Alexander Street Press links
  alman  : Al Manhal
  apb    : Apabi ebooks
  blo    : Bloomsbury Open
  brep   : Brepols (brepolsonline.net)
  crc    : CRCnetBase
  cup    : Cambridge University Press
  ciao   : Columbia International Affairs Online
  cod    : Criterion on Demand
  dgry   : De Gruyter ebook platform
  dram   : DRAM
  duphw  : Duke University Press (via HighWire)
  eai    : Early American Imprints (Readex)
  ebr    : Ebrary links
  ebs    : EBSCOhost ebook collection
  end    : Endeca - Check for undeleted records
  fmgfod : FMG Films on Demand
       knv       : Knovel
      ieee   : IEEE
  igi    : IGI Global
  kan    : Kanopy Streaming Video
  lion   : LIterature ONline (Proquest)
```

Figure 1: Script running in my terminal showing platform list

```
elsif package == "igi"
  sleeptime = 10
  if  page.match(/title="Owned"/)
      access = "owned"
  elsif page.include?("Institution Prices")
      access = "not owned"
  else
      access = "check manually"
  end
```

Figure 3: Code I created for the IGI Global platform

```
elsif package == "asp"
  sleeptime = 1
  if page.include?("Page Not Found")
    access = "Page not found"
  elsif page.include?("Trial login | Alexander Street")
    access = "Trial"
  elsif page.include?("<span>Your institution does not have access to this particular content.</span>")
    access = "Institution does not have access"
  elsif page.match(/<source src="http:\/\/alexstreet\.vo\.llnwd\.net/)
    access = "Video access"
  elsif page.match(/link rel="preconnect"/)
    access = "Video access"
  elsif page.include?("Browse")
    access = "Full access"
  elsif page.match(/(\s|"|'|\/|\.)[Ee]rror/)
    access = "Error returned"
  else
    access = "Check access manually"
  end
```

Figure 2: Code depicting the criteria used for assessing if a library has access to an item on the Alexander Street Press platform

Figure 4: Using Inspect Element to find code in a platform to use in the Access Checker

MARCDownloader

UMD Libraries uses WorldShare Collection Manager to administer and provide access to its e-resources through WorldCat Discovery (WCD) and the OCLC link resolver. While most major collections are represented in the WorldCat knowledgebase (WCKB), not all databases are represented, and the metadata may not be complete. Because the UMD Libraries manages e-resources in the WCKB only, it is critical to ensure that the holdings are fully represented with the metadata needed to facilitate access.

When a WCKB collection does not exist for a platform, there are a couple of avenues you can take to create one: find and transform vendor MARC records into a KBART file (preferably using a tool such as MarcEdit's MARC 2 Kbart Converter) or manually create the collection. These strategies do not work in some situations such as with large databases or if records are not available. The ProQuest Dissertations and Theses Global (PQDT Global) database has been a pain point. The database is very large; as of this writing, it contains about 5 million theses and dissertations.[3] During a study of canceled interlibrary loan borrowing requests, Hilary Thompson found that 16% of them (the largest of any single platform) at the UMD Libraries were for materials available in PQDT Global, but that were not accessible in WCD.[4] There are an estimated 1 million records for items in PQDT Global in WorldCat, but without a corresponding collection, no one can provide access to them within WCD.

MARCDownloader[5] is a Python script I developed that uses the WorldCat Search API to find a record in WorldCat and transform the MARC21XML metadata returned by the API into a KBART file to upload into Collection Manager. The script was initially developed for finding and adding titles in PQDT Global, but I am able to use it for a variety of platforms.

First iteration—Initially, I struggled to create a script that could convert the MARC21XML returned by the WorldCat Search API into a KBART file because I had no experience with XPath to parse and use that metadata. Thus, the first iteration of the script searched WorldCat and wrote the resulting data to a text file; then, I used MarcEdit's MARC 2 Kbart Converter to transform the MARC21XML into the tabular KBART format needed to upload into Collection Manager. To prepare for using MarcEdit, I used a text editor and regular expressions to remove the URLs I did not need and clean up the URLs I did need. Regular expressions were fast, but the time needed to ensure that the data was ready for transformation was prohibitive.

Second iteration—In order to circumvent the manual cleanup, I began exploring using XPath to read and use the MARC21XML returned by the API. After learning XPath, I was able to pull metadata from the MARC21XML and create a KBART file using that metadata. Figure 5 shows the code I use to find title and URL information. This code looks for the subfields A and B in the 245 field, saving the metadata (if available) to the "titleMain" and "subTitle" variables. The script then concatenates those variables to create the "completeTitle," which is then used to write the data to the KBART file. The last line of code finds the URLs in the record (all instances of 856$u), which is parsed by the script to find the desired URLs and then scrubs the string to ensure it is ready to be written to the file.

```
titleMain = record.find("./marc:datafield[@tag='245']/marc:subfield[@code='a']",ns).text
subTitle = record.find("./marc:datafield[@tag='245']/marc:subfield[@code='b']", ns)
if subTitle is None:
    completeTitle = re.sub(r' /','',    titleMain)
else:
    completeTitle = titleMain + subTitle.text
    completeTitle =  re.sub(r' /','', completeTitle)
Urls = record.findall("./marc:datafield[@tag='856']/marc:subfield[@code='u']", ns)
```

Figure 5: Code using XPath to find and extract data from the MARC21XML returned by the WorldCat Search API

```
for Url in Urls:
    Url = Url.text
    print(Url)
    if UrlInput == "PQ":
        docViewUrl = re.search(r'docview/(\d+)', Url)
        if docViewUrl is None:
            gatewayUrl = re.search(r'(openurl.+)', Url)
            if gatewayUrl is None:
                umiUrl1 = re.search(r'.*fullcite/(\d+)', Url)
                if umiUrl1 is None:
                    umiUrl2 = re.search(r'.*(fullcit\?.+)', Url)
                    if umiUrl2 is None:
                        umiUrl3 = re.search(r'.*fullcit/(.+)', Url)
                        if umiUrl3 is None:
                            titleUrl = 'not found'
                        else:
                            titleUrl = 'http://wwwlib.umi.com/dissertations/fullcit/'+umiUrl3.group(1)
                    else:
                        titleUrl = 'http://wwwlib.umi.com/cr/'+umiUrl2.group(1)
                        break
                else:
                    titleUrl = "http://wwwlib.umi.com/dissertations/"+umiURL1.group(1)
                    break
            else:
                titleUrl = "http://gateway.proquest.com/" + gatewayUrl.group(1)
                break
        else:
            titleUrl = "https://search.proquest.com/docview/" + docViewUrl.group(1)
            break
    else:
        if UrlInput in str(Url):
            titleUrl = Url
        else:
            titleUrl = "no match"
```

Figure 6: Code using regular expressions to extract and clean up URLs

```
libr1tm221849:MARCDownloader bbradle1$ python PQDownloader.py
Please input the year for your query:
Enter PQ for PQDT project query or enter custom Url: hdi.loc.gov/loc.mbrsrs/poetry
arch
Checked for English items only?: no
search for dissertations? no
Check for ebooks?(yes/no): no
File name for output: example.txt
Input Collection Name: exampleCollection
Input collection ID: exampleCollectionId
max = 215
(1, 215)
New value of beginPoint is: 1
Getting Marc Data!
1
http://hdi.loc.gov/loc.mbrsrs/poetryarch.94838399
2
http://hdi.loc.gov/loc.mbrsrs/poetryarch.94838530
3
http://hdi.loc.gov/loc.mbrsrs/poetryarch.95770525
```

Figure 7: Screenshot of the script finding records for materials in the Archive of Recorded Poetry and Literature at the Library of Congress

In Figure 6, I have five "if" statements that check the URLs to see if they match any of the identified patterns. If they do, the variable "titleURL" is assigned the corresponding value (see the "else" statements). Using XPath to extract the metadata, the script now can create a KBART file, which can be uploaded (see Figure 7).

KBQuery

When evaluating titles for renewals, cancellations, and other considerations, the personnel in my department often manually search our knowledgebase to collect holdings information. I was approached by a librarian in my department about the possibility of automating the process. Building on my experience using OCLC APIs, I realized I could use the WCKB API to automate the searching and then develop a report that could be useful for a variety of needs as an output.

KBQuery[6] is the script I developed to execute batch searches using the WCKB API. It searches each term, provided by a text file created by the user, and writes the returned metadata to a separate file (including title, OCLC number, coverage date information, and the WCKB collection name). Additionally, it uses the OCLC License Manager API to add perpetual access rights and archival copyrights information to the report, enabling the user to identify all the different collections and platforms the library has for a title. For example, you may subscribe to a journal and have some coverage in an aggregator database, and some years might be available in an OA collection. The report collects that data, enabling the librarian to see where there is overlap and the kind of access that is available.

The script currently supports two different functions: checking a particular collection to ensure that entitlements are selected and finding all the collections a title has been selected in for WCKB. After discussions with a librarian who is interested in the script, I am working on developing a feature to enable the user to list a set of knowledgebase collections to limit the script to those collections (see Figure 8).

```
libr1tm221049:KBQuery bbradle1$ python kbquery.py
Input File: example.txt
Name of file to save results: exampleOUTPUT.txt
What are you searching on? ISSN, ISBN, OCN, or Title: OCN
To search in a particular collection enter its ID. If you want to search a list
of collections, type 'list'. Type 'no' otherwise: no
```

Figure 8: KBQuery starts with several prompts when the script starts.

The report contains 11 fields, some of which are standard metadata, such as title, standard number, and OCLC number (see Figure 9).

The first field in the report—number—uses a "#.#" pattern. The first number corresponds to the search term, and the second corresponds to the item's place in the search results. For example, 1.1 is the first search term and the first matching result, whereas 1.3 is the first search term and the third match. The number field enables a librarian to quickly see how many matches there are for a title selected in the WCKB. For instance, the second title is selected in six different collections. During a serials review, a librarian could then look at the collection name, coverage, perpetual access, and archival copy fields to help inform any decision making.

Finally, the status field is largely used in the first use case I previously described. When the search is limited to a particular collection, the script first checks to see if the title has been selected; the WorldCat Knowledge Base API only enables searching for the selected titles. However, if the script does not find a match, it downloads the entire KBART file—which is accessible via a URL provided by the API—and then checks the file to look for a match. If the script finds a match in the complete KBART file, it would then record that the title is unselected. In the second use case, it is only searching for selected titles.

Final Thoughts

The scripts discussed may be helpful to some librarians more than others, but I hope that the pathway I illustrated—from editing and using another librarian's script to creating a script that synthesizes data from multiple sources—shows how learning to code can be a step-by-step, progressive process that not only enriches you personally, but can benefit your library and even the community at large.

Benjamin Bradley
(Bbradle1@umd.edu) is a discovery librarian at the University of Maryland (UMD) Libraries. He has worked with e-resources at the UMD Libraries since 2015 and accepted his current position in 2017.

Endnotes

1. Code available on GitHub: github.com/UNC-Libraries/Access-Checker.
2. Spurgin, Kristina M. (2014). "Getting What We Paid For: A Script to Verify Full Access to E-Resources." *Code4Lib Journal*, 25. journal.code4lib.org/articles/9684.
3. Total number of theses and dissertations taken from the PQDT Global webpage: proquest.com/products-services/pqdtglobal.html.
4. Thompson, Hilary (2015). Find It Fail: What ILL Can Tell Us About Challenges Related to Known Item Discovery. hdl.handle.net/1903/16871.
5. Code available on GitHub: github.com/bradley-benjamin26/WCSearchAPIMARCHarvester.
6. Code available on GitHub: github.com/bradley-benjamin26/KBQuery.

number	status	title	ISBN or ISSN	ocn	Collection Name	KB ID	coverage	Perpetual Ac	Archival Cop	Search Term
1.1	selected	012 IEEE 28th Symposium on N	9781467317450	812607995	IEEE Xplore All Conference Proceedings	11067293	ebook@2012	no or silent	yes	812607995
1.2	selected	012 IEEE 28th Symposium on N	9781467317450	812607995	IEEE/IET Electronic Library (IEL)	no KB ID	ebook@2012	no or silent	yes	812607995
1.3	selected	Mass Storage Systems and Tecl	No Standard nun	812607995	IEEE Proceedings (UMC)	812607995	ebook	no or silent	yes	812607995
2.1	selected	Energy Journal	0195-6574	563150979	Business Source Complete	514992	fulltext@1990-01-01	no or silent	silent	563150979
2.2	selected	Energy Journal	0195-6574	563150979	International Association for Energy Economics	806402	fulltext@1982	no or silent	silent	563150979
2.3	selected	Energy Journal, The	0195-6574	44393051	ISTOR Archive Collections Complete	9332124	fulltext@1980-01-01~P4Y	no or silent	yes	563150979
2.4	selected	Energy Journal	0195-6574	563150979	Academic Search Ultimate (UMC)	customer.12	fulltext@1990-01-01	no license fo	no license fo	563150979
2.5	selected	Energy Journal, The	0195-6574	563150979	Materials Science & Engineering Database	27557894	fulltext@1998-01-01~2013-07-01	no license fo	no license fo	563150979
2.6	selected	The energy journal /Internatior	0195-6574	5585856	UMC Print Journals (UMC)	no KB ID	print@1980	no license fo	no license fo	563150979
3.1	selected	#Democracy : the Internet and	No Standard nun	896155769	Proquest Dissertations and Theses Global (UMC)	896155769	ebook@2014	no license fo	no license fo	896155769
4.1	selected	291	2330-5983	448035264	Blue Mountain Project	448035264	fulltext@1915-04~1915-11	no license fo	no license fo	448035264
4.2	selected	291	1054-7193	448035264	ISTOR Arts & Sciences VIII Collection	3238533	fulltext@1915-03-01~1916-02-01	no or silent	yes	448035264
5.1	selected	1,001 Ways to Get Promoted	9781564144300	44955812	All EBSCO eBooks	251017	ebook@2000	no or silent	silent	44955812
6.1	selected	SYMPOSIUM: CANADIAN JOUR	2154-5278	971917295	Portico journals (UMC)	c1284.73	fulltext@2014~2016	yes	silent	971917295
7.1	selected	European Journal of Endocrino	0804-4643	40801501	BioScientifica	806679	fulltext@1948	no or silent	yes	40801501
7.2	selected	European journal of endocrino	0804-4643	29970781	UMC Print Journals (UMC)	no KB ID	print@1994	no license fo	no license fo	40801501
8.1	selected	Reproduction	1470-1626	60638423	BioScientifica	836325	fulltext@1996	no or silent	yes	60638423

Figure 9: Example of a report created by KBQuery, viewed in Microsoft Excel