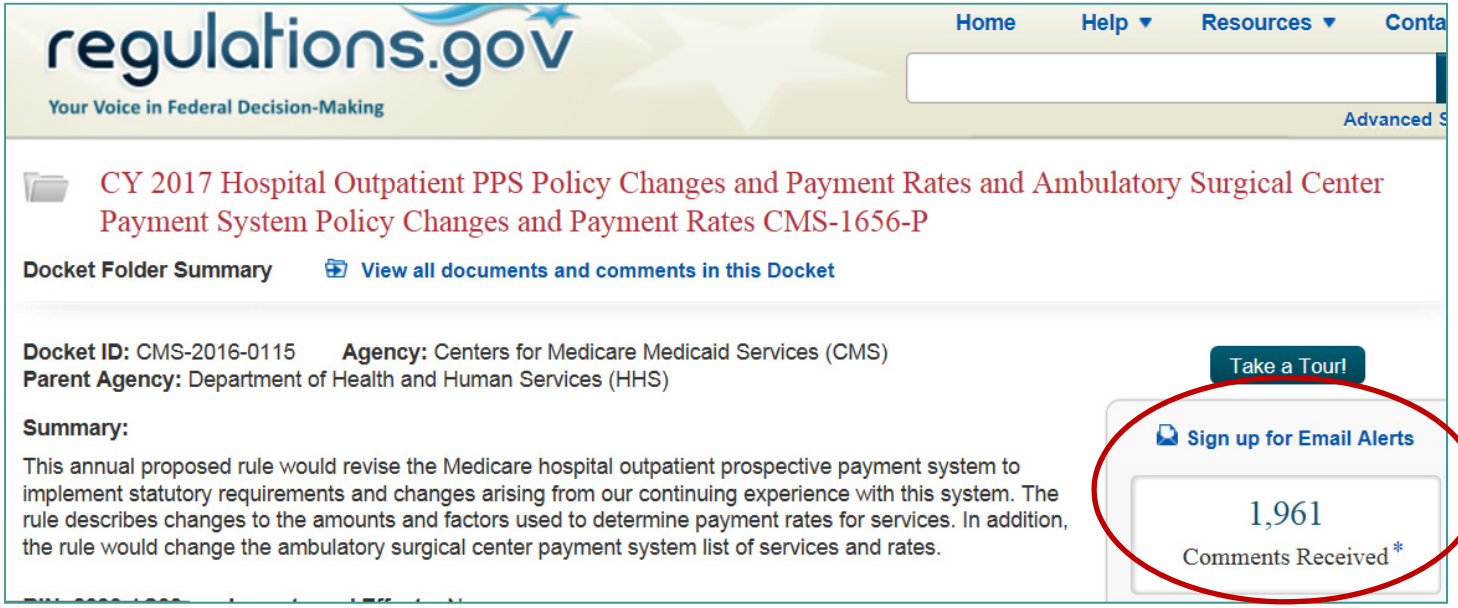
Getting and Analyzing Inconveniently Structured Data in the Internet Era: Making Friends with Python, Webscraping, and APIs



Robert Letzler PhD, Center for Enhanced Analytics, US GAO

Disclaimer: These remarks do not necessarily reflect the views of the US GAO

For more information, contact me: LetzlerR@GAO.gov

Goal: Inform decisions about automating repetitive work that is hard in other tools

Topics of discussion

- **The right tools can tame a deluge of "inconveniently" formatted data**
- Example: FITARA conversion and analysis
- Bringing more automation to an audit organization
- Where to get started and find help

Modern agencies create a deluge of "inconveniently" formatted data

- Government increasingly operates, publishes, discloses, and gets public input electronically
- Data are often fragmented on the web (or a network drive)
- And often stored in a markup language or PDF:
  - Web = HTML
  - Microsoft Office (zipped) XML: DOCX, PPTX, XLSX
  - Data = XML, JSON
- SAS, Stata, Python, R, and Excel can handle "conveniently" formatted data tables

This is an inconvenient package …

until you get a can opener!

Python is a can opener and a lot more

Free, approachable tools can help collect and extract that "inconveniently" formatted data

- The web, markup languages, and PDFs work in predictable, understandable ways
- Tools shine for the big easy: easily described, very repetitive tasks

Sharing experiences from a small 18-month effort with no budget

- A few programming courses and willingness to read, Google, and experiment goes a long way
- Software is free; Skill and organizational issues are tougher.
- Here are examples of data types, extraction challenges, and GAO projects from a handful of people over the last 18 months

Assessing what is involved in automating website interaction: first see if it provides tools for you

Best case: Documentation for "developers" or of the "API"

APIs: user interfaces for computers

Websites have lots of formatting for humans

- An API eliminates this clutter
- A typical API offers:
  - Ability to query through a web request: http://api.data.gov:80/regulations/v3/documents.json?api_key=DEMO_KEY&dktid=OCC-2013-0003
  - Results in JSON – key:value pairs, which map to Python dictionaries

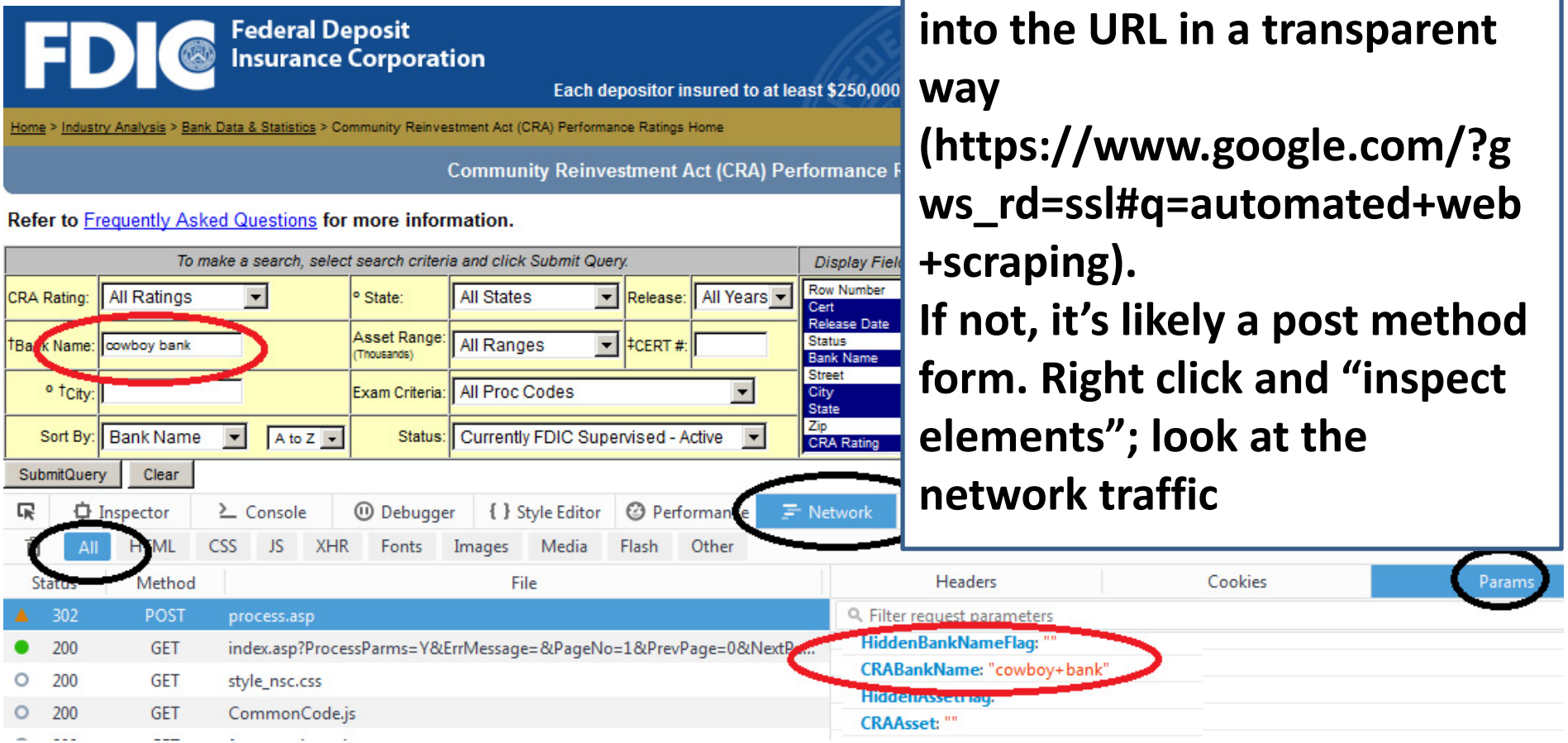

```
{ "documents": [
   { "agencyAcronym": "OCC",
   "allowLateComment": false,
   "attachmentCount": 1,
```

GAO projects using APIs

- Downloaded 190 public comments on the Community Reinvestment Act from regulations.gov
- Generalizing that code to create an in-house web-form to generate a ZIP file containing the documents on any docket
- Downloading hundreds of proposed rules listed on team spreadsheets from FederalRegister.gov
- Building a database of who is coauthoring or citing each other's work in green chemistry using Elsevier SCOPUS

If there's no API, see how the site requests or presents information

First see if the query is going into the URL in a transparent way (https://www.google.com/?gws_rd=ssl#q=automated+web+scraping).
If not, it's likely a post method form. Right click and "inspect elements"; look at the network traffic



You can typically automate using just a few relevant pieces – e.g. post-method form submission and page requests; or tags around relevant elements

A GAO project involving post-method forms

- We needed to identify FDIC Community Reinvestment Act exams conducted in 2015
  - This required running ~300 queries (50+ jurisdictions x 2 year x 3 exam types); checking for 2015 in the URL; downloading the 2015 PDFs and creating a comma separated value (CSV) data table. We analyzed the table in SAS
- Website was complicated but limited; the substance complicated; and the audit needs evolving. The project took us about a month

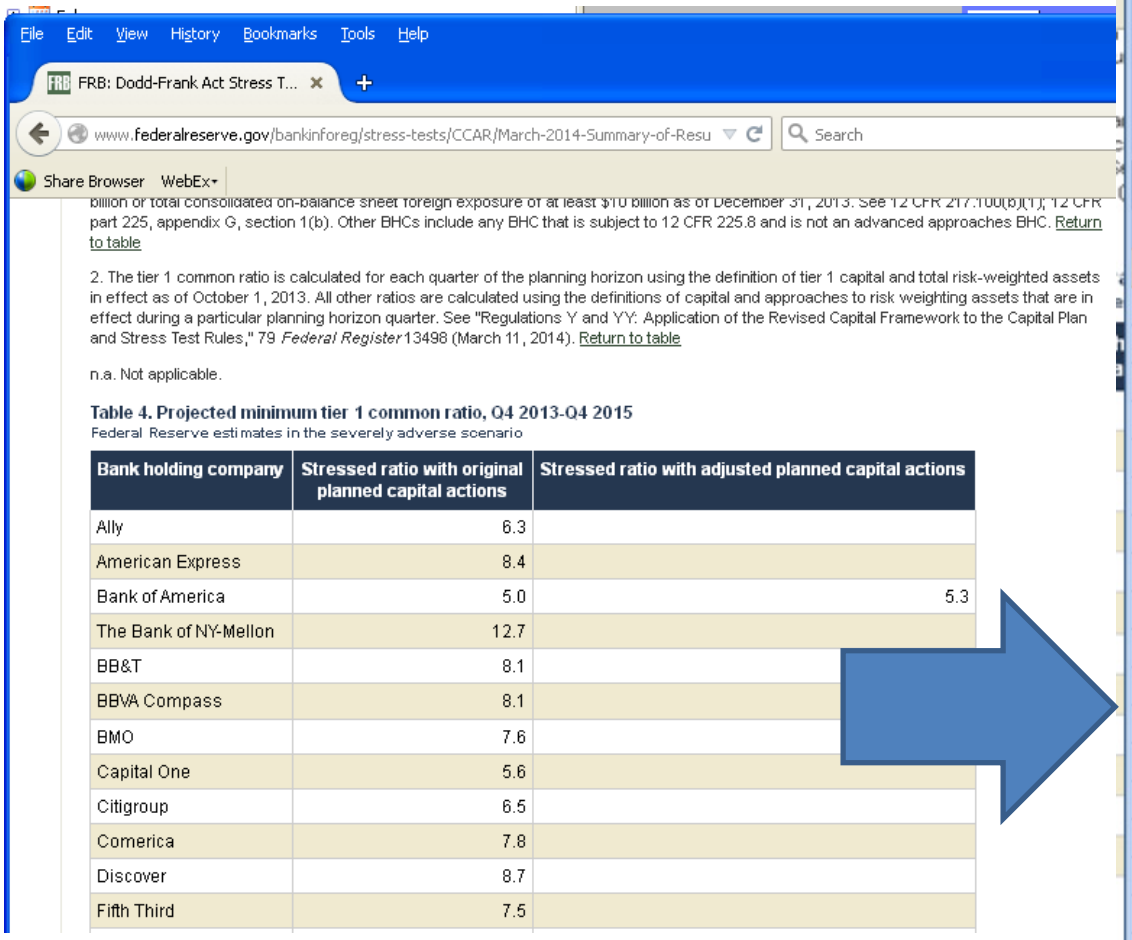
Webpages are written in a markup language, HTML (typically plus JavaScript and Cascading Style Sheets)

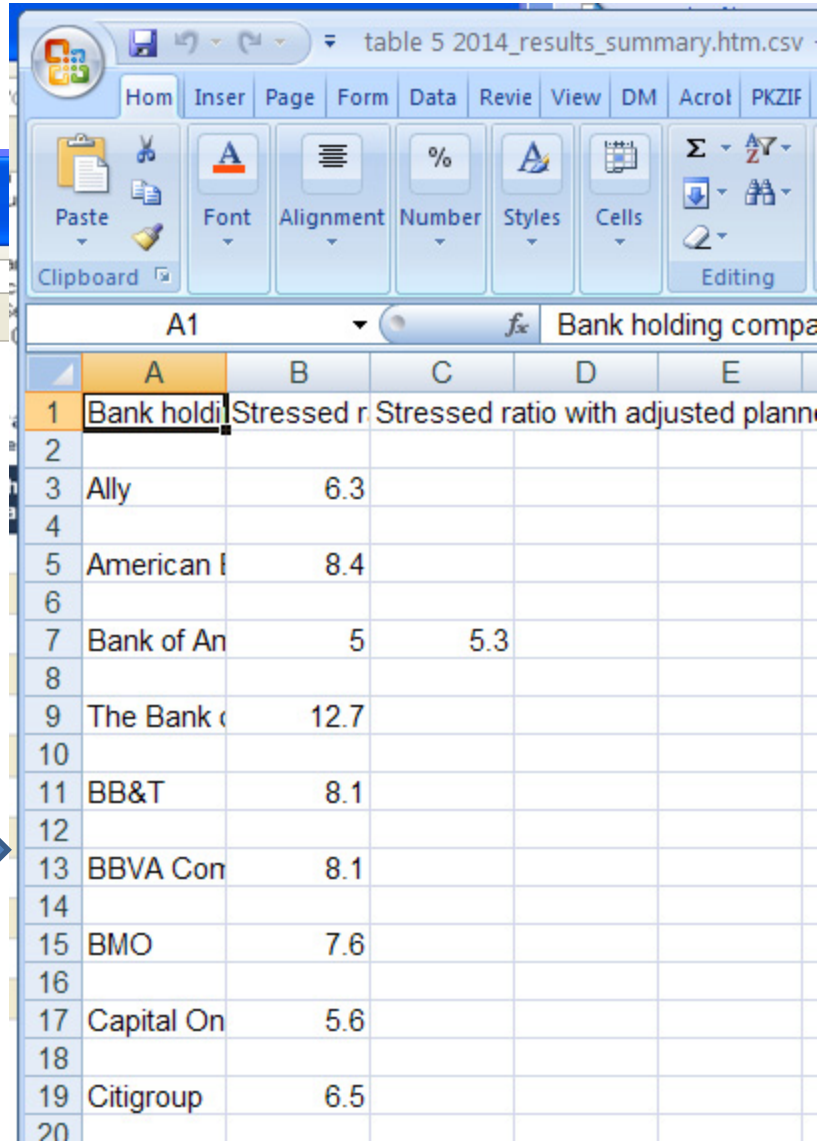
- Use “view source” to look at the HTML

East	$20,000
West	$49,950

- <TABLE> <TR><TD> East</TD><TD> $20,000</TD></TR>
- <TR><TD>West</TD><TD> $49,950</TD></TR></TABLE>
- If we extact the contents of all the <TD>table cells</TD> and save them in comma separated form, the results will go right into Excel, SAS, or Stata

Extracting data tables from webpages





Fed data were in a fragmented, inconvenient format

We need to import them in bulk

(can process data in Python, SAS, Stata, or Excel)

A documented process is desirable

Web site automation difficulty

Difficulty	Type	Strategy
Easiest	Static sites	just grab the relevant webpages (a copier like WinHTTrack may be sufficient)
Easy	Get method forms – everything important is in the URL:	Generate desired HTTP requests (e.g. https://www.google.com/?gws_rd=ssl#q=automating+HTTP+requests) and use results
Easy	API –a user interface tuned for computers	Generate HTTP requests to run desired queries; interpret the conveniently formatted results
Not so hard	Post method forms	Inspect element, watch the network to identify the "payload" to send the form and additional requests to make
Involved	+Javascript, Cookies, CSS, AJAX, poor organization, anti	Above strategies plus additional steps

Information on computer files is readily accessible

- Data elements available
  - File names – including extension
  - Directory structures
  - File size and date
- Operations available:
  - Selecting files to work with based on that data
  - Copying, moving, deleting, creating, aggregating
  - Recording these data for analysis, inventory

Python's powerful file management capabilities are part of many of our projects

- Copying and renaming sampled files to a "sample" directory
- Performing the same operation on all the PDF files in a specified directory
- Adding to and extracting from ZIP archives
- Transcribing directory information into Excel
- Checking for compliance with policies about the use of transfer folders

PDF is built to describe page layout, not meaning

- APIs are often an ideal data source. PDF is never the ideal data source – but it may be the best available
- It may contain (high quality or garbled) machine readable text; OCR can make images into text.
- Extracting and processing plain text is often straightforward
- Initial attempts to extract tabular data using specialized software yielded mixed results

A GAO audit automated search and documentation of compliant language in PDFs

- Goals:
  - Checking ~200 messy PDF files to see if each contract had 3 required, boilerplate clauses
    - Edit distance was a crucial tool to identify the best match
  - Rapidly find apparent, obvious non compliance for further investigation (e.g. agency sent wrong doc)
  - Extract the PDF pages with interesting language
  - Allow one analyst to efficiently verify rather than one to find and document and one to verify

Automatically filled much of the DCI

Contract Name	DM link to file containing pages with match for terrorism clause	match location(s) for terrorism clause in the file with the best match	OCR text of potential match for terrorism clause	analyst review and notes
Chemtronics	DM#123	Page 2, top	prohibitlon against Sueport for Terronsm: (a) The Contrador/Reclplent …..	
IBEX	DM#124	Page 12 middle	frohibilion against Support for Terrorism: (a) The Contractor/Recipient ….	
Research Octagon Institute	DM#125	Page 4	Prohibition against Support for Terrorism: (a) The Contractor/Recipient …	

Frontiers

- Named Entity Recognition
  - Automatically extracts many of the names, locations, and organizations in plain text
- Text classification
  - Show the computer example documents that are or are not something of interest (a discussion of an IT system intrusion). Then have the computer find more examples of interest

We tackled the projects discussed here in Python

Python is often the right tool

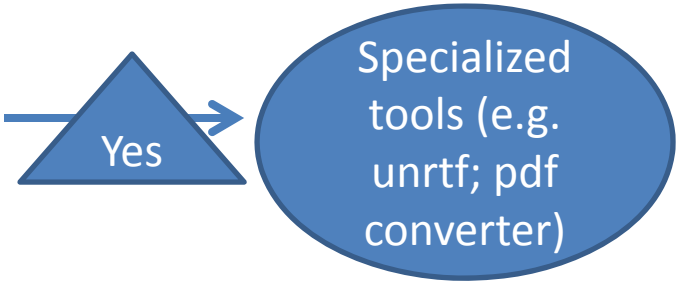
- A popular, high level (i.e. a little code can do a lot) programming language designed to be easy to learn
- Python and hundreds of libraries are free
- Multiplatform (Windows, UNIX, Mac)
- Supports ideas you've learned in SAS, Stata, or Java
- Extremely flexible: Many applications beyond data collection / extraction
- Other languages' capabilities overlap Python's including C/C++/Java/C#, PERL, R, SAS, Stata

Is Python right for you?

- Is the task just transforming or analyzing an existing database

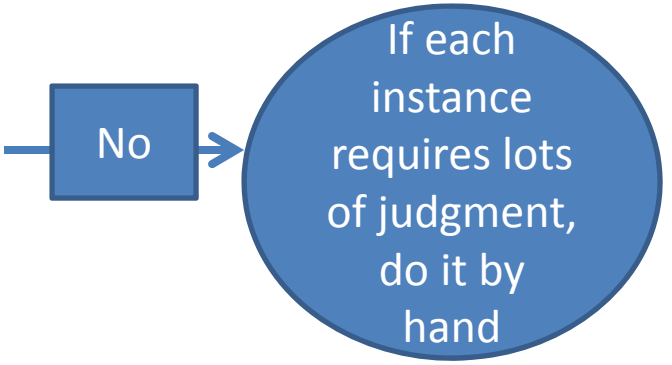
No

- Do we have / can we justify a specialized tool that is faster or deals with any nasty complexities?



No

- Can you write a precise "pseudo-code" recipe for the task?



Yes

Consider Python!

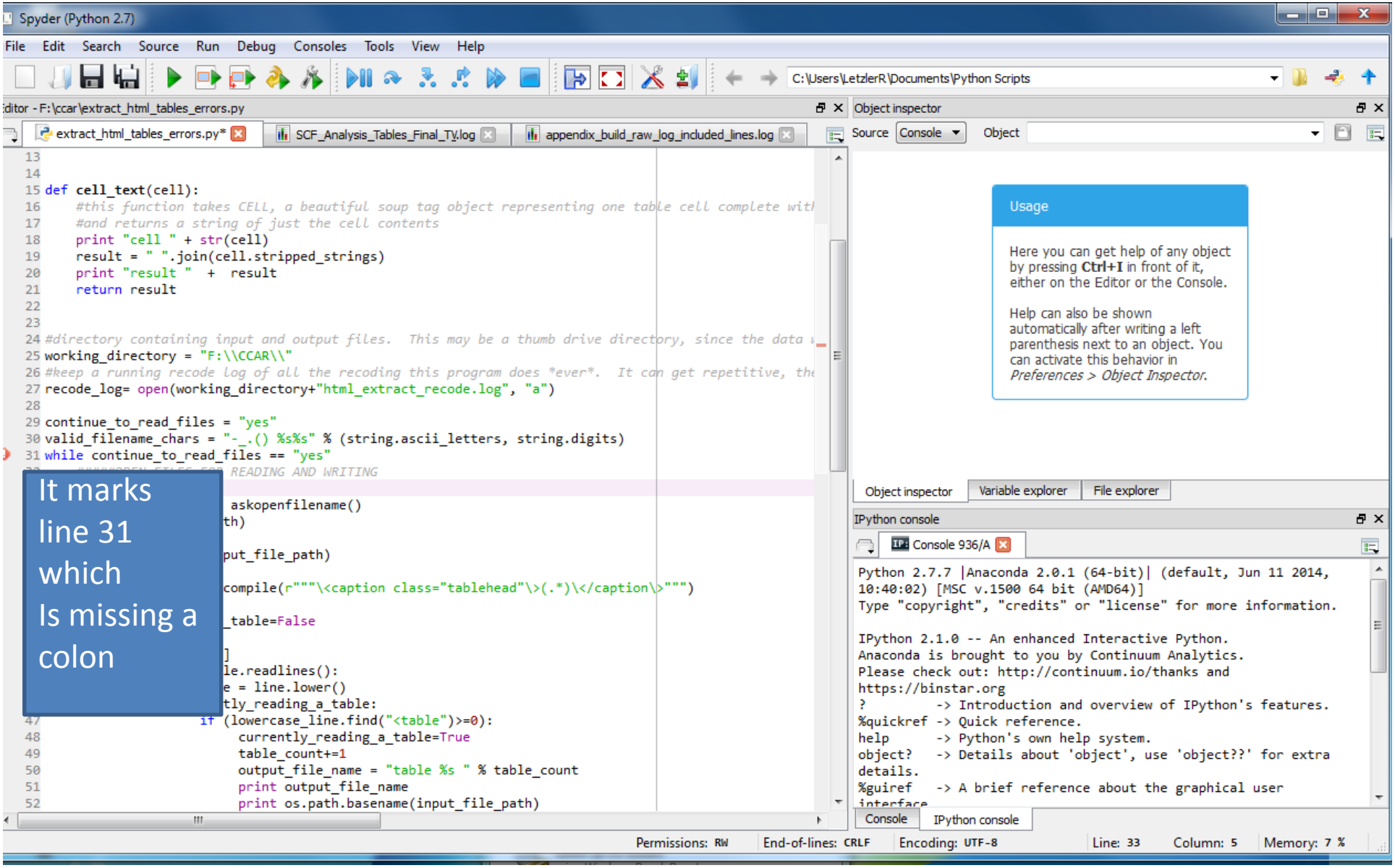
- “Python is still my favorite language for making my computer do things. …C# is my favorite language for building systems.”

  -professional software developer

Putting Python in Context

- SAS and Stata are at their best working with databases
- Python is a general purpose language with database, file, text, math, and internet libraries available
  - Far more flexible; more varied uses
  - It assumes less, so you'll have to write a bit more

Integrated development environments (e.g. Spyder, IDLE) help write, **debug**, and **run** (press F5) Python



Goal: Inform decisions about automating repetitive work that is hard in other tools

Topics of discussion

- The right tools can tame a deluge of "inconveniently" formatted data
- **Example: FITARA conversion and analysis**
- Bringing more automation to an audit organization
- Where to get started and find help

FITARA JSON conversion: the ask

- Agencies posted updates to their FITARA implementation plans on April 30, 2016 at https://management.cio.gov/plans/"
- Step 1: convert the plans from JSON to Excel
- Step 2: find instances where "milestoneStatus" = "In progress" and "MilestoneTargetCompletionDate" is before the file date
- As usual, the most time consuming part will be dealing with non standard formats or unexpected entries. That is omitted for brevity here – the demo code leaves out six agencies that use non standard file formats.

Strategy: convert JSON to pipe delimited text, which Excel reads

{"milestones": [

{ "milestoneID": 1,

"milestoneDesc": "The Enterprise Information Technology (IT)…. ",

"milestoneTargetCompletionDate": "2015/12/31",

"milestoneStatus": "Complete",

"milestoneStatusDesc": "Completed on 2015/12/03. …",

"commonBaselineArea": "budgetFormulation", "dcoiArea": "nonDataCenter" },

A python dictionary is denoted: {“key”:value,”key2”:value}

1|The Enterprise Information Technology (IT) …. |2015-12-31 00:00:00|Complete|Completed on 2015/12/03…. |budgetFormulation|nonDataCenter

Python code

- Is extremely specific machine readable instructions
- Mixes (clever) application of simple tools [e.g. loops, lists, variables, dictionaries] and (simple) application of specialized libraries [e.g. requests, trace]
- Generally has only as much common sense as you give it (if 2000/1/1 is the same as Jan-1-00, say so)

Define some key lists and a function

the requests library facilitates http:// requests

```
#requests has routines for accessing the web through HTTP requests
#we'll use datetime to convert text dates into numbers we can readily compare.
import requests, datetime, csv,trace, sys

#this definition is a JSON dictionary lookup that returns a blank if the key does
not exist in the dictionary
#and eliminates stray newlines
def extract(json_dict,key):
    if key in json_dict:
        return str(json_dict[key]).replace("\n"," ")
    else:
        return ""

def main(working_directory):
    #List of links captured from the "Public FITARA April 30th Milestone Updates"
section of https://management.cio.gov/plans/
    #some agency's links were missing:  DoD, Energy and Labor

    fitara_link_list =
["http://www.usda.gov/digitalstrategy/fitaramilestones.json",
"https://www.commerce.gov/sites/commerce.gov/files/fitaramilestones.json", …
```

A python list is denoted [item1,item2,item3]

Open files for output and put headers on them

```
all_agency_file = open(working_directory+"Fitara.txt","w",
errors="backslashreplace" )
all_agency_csv = csv.writer(all_agency_file, lineterminator='\n', delimiter="|")

all_agency_csv.writerow(["URL","file_date","milestoneID","milestoneDesc","miles
toneTargetCompletionDate","milestoneStatus","milestoneStatusDesc","commonB
aselineArea","dcoiArea"])


overdue_file = open(working_directory+"Fitara_inProgress_overdue.txt","w",
errors="backslashreplace")
overdue_csv = csv.writer(overdue_file, lineterminator='\n', delimiter="|")

overdue_csv.writerow(["URL","file_date","milestoneID","milestoneDesc","milesto
neTargetCompletionDate","milestoneStatus","milestoneStatusDesc","commonBas
elineArea","dcoiArea"])
```

Visit each agency's URL and get its update date

```
for agency_URL in fitara_link_list:
        print(agency_URL)
        agency_plan =  requests.get(agency_URL)
        file_date =
datetime.datetime.strptime(agency_plan.json()[
"updatedDate"],"%Y/%m/%d")
```

Take steps for each milestone– extract target dates, create a list of fields, write it to the appropriate files

Request_response.json() maps the JSON to a Python “dictionary.”

```
for  milestone in agency_plan.json()["milestones"]:
        targetCompletionDate =
datetime.datetime.strptime(milestone["milestoneTargetCompletionDa
te"],"%Y/%m/%d")
        output_list = [agency_URL, str(file_date),
extract(milestone,"milestoneID"), extract(milestone,"milestoneDesc"),
str(targetCompletionDate), extract(milestone,"milestoneStatus"),
extract(milestone,"milestoneStatusDesc"),
extract(milestone,"commonBaselineArea"),
extract(milestone,"dcoiArea")]
        all_agency_csv.writerow( output_list)
        if (milestone["milestoneStatus"]=="InProgress") and
(targetCompletionDate < file_date):
            overdue_csv.writerow(output_list)
```

Close the files and create an audit trail

```
    all_agency_file.close()
    overdue_file.close()
    #PROBABLY WE SHOULD ADD A LOG THAT RECORDS THE FILE
DATE AND SIZE OF THIS FILE; THE SIZES OF ALL THE
DOWNLOADED FILES, THE START TIME, ETC.

 # create a Trace object -- which will create a log file
that counts the number of executions of each line below.
tracer = trace.Trace(
     #the goal of this line -- which comes straight from
the sample code -- is to generate trace files only for
GAO-written code and not more than a dozen trace files for
Python-supplied code
     ignoredirs=[sys.prefix],
     trace=0,
     count=1)
```

Run everything and write out the trace log file

```
working_directory = "R:\\letzlerr\\FITARA\\"

# run the whole above program while using the tracer object to log which lines got executed.  This is separate from "logging," the file I/O log above
tracer.run('main(working_directory)')
#now write the trace results to disk
trace_results = tracer.results()
trace_results.write_results(show_missing=True, coverdir=working_directory)
```

Goal: Inform decisions about automating repetitive work that is hard in other tools

Topics of discussion

- The right tools can tame a deluge of "inconveniently" formatted data
- Example: FITARA conversion and analysis
- **Bringing more automation to an audit organization**
- Where to get started and find help

If you are writing code for a single data set, you don't care about weird circumstances that don't arise in it

- But if the code needs to deal with challenges unseen in the test data, need to design for those possibilities.
- Auditors often write code for one data set; Google engineers typically deal with more general challenges – which are harder, more technically interesting

Asserting that things are as expected lets Python warn you if they're not

- Assert checks a condition you specify and raises an "exception" if it is not true.
  - SSN length? Thou shall count to 9; 10 is right out!
  - Assert is your friend.
- In Python, an exception can
  - stop the program and issue an informative error
  - or jump to an "except:" section of the code

Python facilitates documenting your work

#anything after a pound sign is a comment

- **trace** library creates a log showing how many times each line of code executed.
- Can create your own log files documenting what you did, names, sizes, and dates of files you created, etc.
- Not as automatic as SAS or Stata logging, but you can build exactly the documentation your reviewer needs.

GAO uses internal guidance papers to ensure appropriate planning, audit trail, and review of computer code

- Planning data analysis
  - A process we harness to coordinate between audit-specific subject matter experts and technical experts
- Documenting code
- Review and verification of code
- Same general guidance papers we use for SAS are appropriate for Python

Building institutional comfort will require effort

- Write a plain English record of analysis (ROA) describing approach and results; have a technical colleague review the ROA and the code; then less technical colleague can review the final report that uses the ROA as support
- We've started carefully; expanding gradually
  - First automating things we have traditionally done by hand
  - Building confidence through considerable human review

The right role for Python depends on your agency's needs and existing tools

- GAO has significant investments in SAS and Stata and accompanying skills.
  - Because of that, Python and R are competing to be the go-to tools for challenges that break the SAS and Stata database-tables-in, statistics-or-tables-out mold
- The need for enough users to do meaningful internal peer review suggests building depth in a few tools
- What is the minimum viable number of users in your organization?

Relevant and transferrable skills to look for on resumes and in existing staff

- Programming training and experience translates to Python:
  - Relevant Languages: Python, C++, C#, C, Java, etc.
  - A few semesters of computer science teach useful ideas.
    - These courses readily available online
  - Web development and database experience – Javascript, Cascading Style Sheets (CSS), SQL, etc. – helps understand context
  - Seek: some basic coding skills, a logical mind, attention to detail, and a willingness to learn
- Other statistical tools:
  - R is fairly similar to Python
  - SAS and Stata use some similar skills, but Python has some additional concepts

Finding the right people

- Good experiences with interns – one working on a masters in CS & Policy; the other a straight MPP who learned to code in his stats/math/psych research past and was willing to learn
- Opportunity seems to be people who care about audit mission who have some coding skills.
- Audit specific coding challenges unlikely to attract great software engineers
  - We write code memos; they write code novels

Communication and coordination are important challenges

- Mission and technology people both have complex expertise, and specialized vocabulary – easy to talk past each other
- Develop translators who are comfortable and skeptical in both languages
  - Intuition for what kind of data exist where at agencies and what's involved in extracting it is valuable
- GAO rule of thumb is that we spend 80-90% of data analysis time cleaning and preparing data. Non programmers misunderstand this

Goal: Inform decisions about automating repetitive work that is hard in other tools

Topics of discussion

- The right tools can tame a deluge of "inconveniently" formatted data
- Example: FITARA conversion and analysis
- Bringing more automation to an audit organization
- **Where to get started and find help**

Getting Python

Python is free, open source software (by nerds, for nerds)

Python alone: available from Python.org;

Python is likely already installed on Linux/Mac computers, maybe even on Windows

(It's got plenty of system administration capabilities that make it attractive to IT professionals)

Python distributions make life better

Python distributions with the Numeric/Scientific Python Stack have two advantages:

Technical: The versions of the libraries and Python work together

Bureaucratic: One rather than many installations

GAO uses Anaconda – which is – to our knowledge --the only free, multiplatform Numeric Python stack that supports Python 3.x. It comes with 180+ libraries

Distribution options here: https://www.scipy.org/install.html

The easy way to get Python into your agency may be to install it outside the main network

- Python came to GAO on computers outside of our main network – including some not networked at all.
- Strong internal controls on software installation on the main network are common and reasonable
- We were building the case to roll it out more broadly when our IT department decided to install Python for IT's own purposes

Python 2.x and Python 3.x are (slightly) incompatible

- GAO ARM/CEA has switched to Python 3; few complaints
- Python 3 reduces pitfalls and confusion
- *"Python 2.x is legacy, Python 3.x is the present and future of the language"*
- **Possible to write code that works in Python 2.7 and Python 3.x.**
- If you have a Linux/UNIX/Mac computer, it likely has Python 2.x installed
- Some training material still in Python 2

Python 3 is better designed than Python 2

- Backstory: Python 2.x included some questionable or dated decisions e.g.:
  - 5.0/4.0 = 1.25 but 5/4 = 1
  - ASCII text (1963 technology) rather than Unicode (supports Chinese characters and emoticons ☺)
  - Python 3 defaults to handling Unicode errors by throwing program-terminating exceptions; you'll likely want to set it to backslash replacing instead
  - **In 2008, Python 3 fixed them; required updating many libraries to restore compatibility. Python 2.x lives on.**
- Python 2 and Python 3 can coexist on the same system; Anaconda has nice "virtual environment" tools to facilitate this

Don’t be afraid:
Plenty of high quality help is available

Start on Google or Python.org; then escalate

- Python tutorial; online courses
- Python help / **docs.python.org/ scipy.org**
- Discussion boards at e.g. stack overflow

Google searches all of these at once

- Colleagues who program in Python or other languages (many ideas, pitfalls, and approaches are the same)
- Post to StackOverflow
- GAO has a support contract

I work in a new GAO center and help coordinate a new GAO community of practice

GAO Center for Enhanced Analytics goals:
-enhance access to data sources
-assess, customize, and help deploy new technologies
-promote novel analytic approaches
-strengthen analytical skills

The Data, Tools, and Analytics (DaTA) Community of Practice: staff agency wide interested in efficient, reliable and insightful data analysis.

Thank You!

If you have further questions, please contact me at:

letzlerr@gao.gov

www.gao.gov

ADDITIONAL RESOURCES

Python has specific libraries for each example

Dialog boxes:

Tkinter is Python's de-facto standard graphical user interface

Text processing:

string (character strings)

re (regular expressions)

Batch querying the web

urllib (lets you read webpages just like files)

also: *bs4 (*Beautiful soup; an HTML parser)

Deleting outdated files/identifying file management policy violations

os (operating system)

Custom calculations: optimization, simulation

numpy/*scipy* (Numeric Python, Scientific Python)

Surveys/lookup: web access to database

django (Django, “The Web framework for perfectionists with deadlines”)

Analyzing a database

CSV (facilitates work with comma separated value text files)

pandas (data structures and data analysis tools)

Not to mention Pyrex and GrumPy