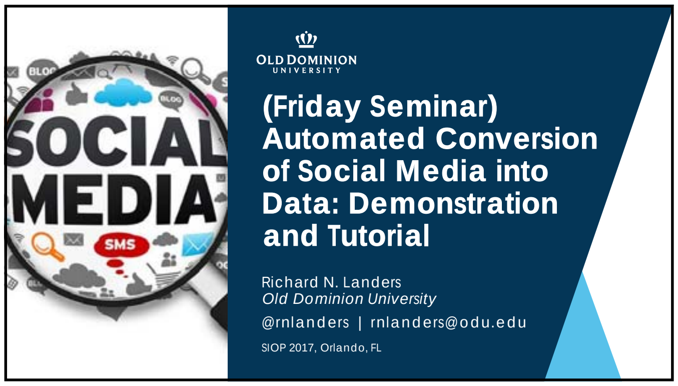
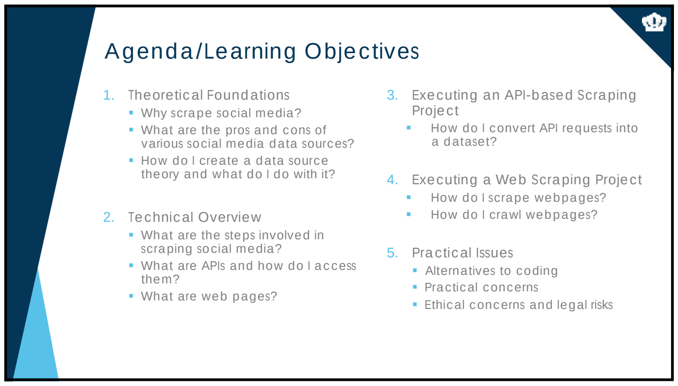


(Friday Seminar) Automated Conversion of Social Media into Data: Demonstration and Tutorial

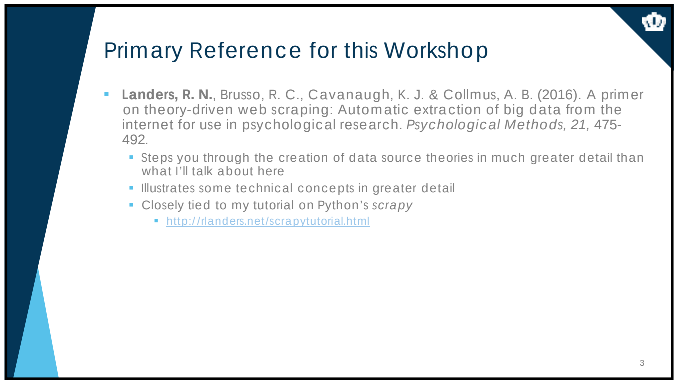
Richard N. Landers
Old Dominion University
@rnlanders | rnlanders@odu.edu

SIOP 2017, Orlando, FL



Agenda/Learning Objectives

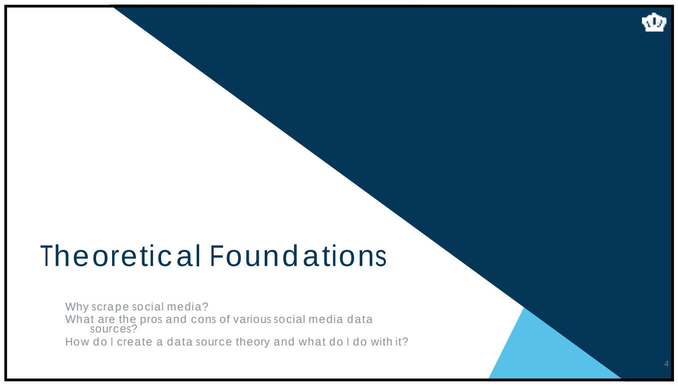
1. Theoretical Foundations
   - Why scrape social media?
   - What are the pros and cons of various social media data sources?
   - How do I create a data source theory and what do I do with it?
2. Technical Overview
   - What are the steps involved in scraping social media?
   - What are APIs and how do I access them?
   - What are web pages?
3. Executing an API-based Scraping Project
   - How do I convert API requests into a dataset?
4. Executing a Web Scraping Project
   - How do I scrape webpages?
   - How do I crawl webpages?
5. Practical Issues
   - Alternatives to coding
   - Practical concerns
   - Ethical concerns and legal risks



Primary Reference for this Workshop

- **Landers, R. N.**, Brusso, R. C., Cavanaugh, K. J. & Collmus, A. B. (2016). A primer on theory-driven web scraping: Automatic extraction of big data from the internet for use in psychological research. *Psychological Methods, 21,* 475-492.
  - Steps you through the creation of data source theories in much greater detail than what I'll talk about here
  - Illustrates some technical concepts in greater detail
  - Closely tied to my tutorial on Python's *scrapy*
    - http://rlanders.net/scrapytutorial.html

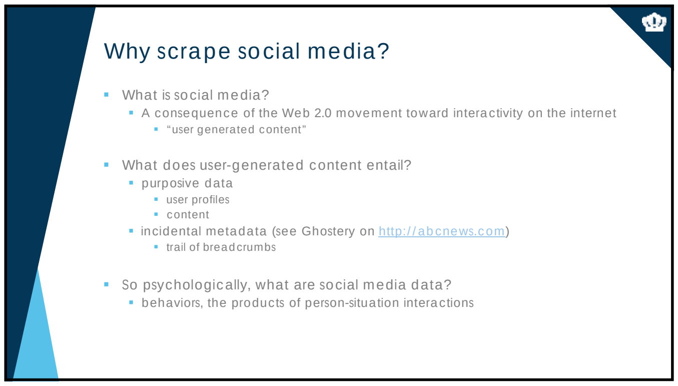
3



Theoretical Foundations

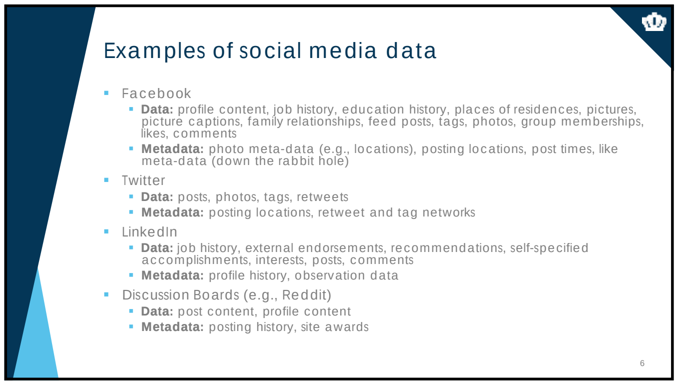
Why scrape social media?
What are the pros and cons of various social media data sources?
How do I create a data source theory and what do I do with it?

4



Why scrape social media?

- What is social media?
  - A consequence of the Web 2.0 movement toward interactivity on the internet
    - "user generated content"
- What does user-generated content entail?
  - purposive data
    - user profiles
    - content
  - incidental metadata (see Ghostery on http://abcnews.com)
    - trail of breadcrumbs
- So psychologically, what are social media data?
  - behaviors, the products of person-situation interactions



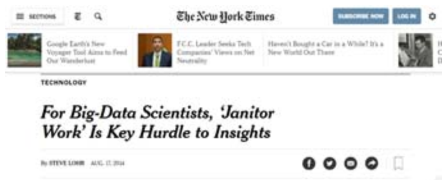
Examples of social media data

- Facebook
  - **Data:** profile content, job history, education history, places of residences, pictures, picture captions, family relationships, feed posts, tags, photos, group memberships, likes, comments
  - **Metadata:** photo meta-data (e.g., locations), posting locations, post times, like meta-data (down the rabbit hole)
- Twitter
  - **Data:** posts, photos, tags, retweets
  - **Metadata:** posting locations, retweet and tag networks
- LinkedIn
  - **Data:** job history, external endorsements, recommendations, self-specified accomplishments, interests, posts, comments
  - **Metadata:** profile history, observation data
- Discussion Boards (e.g., Reddit)
  - **Data:** post content, profile content
  - **Metadata:** posting history, site awards

6

So what can I do with web scraping?

- The first step of “big data science,” data wrangling/munging



TECHNOLOGY

For Big-Data Scientists, ‘Janitor Work’ Is Key Hurdle to Insights

By STEVE LOHR AUG. 17, 2014

SECTIONS
The New York Times
SUBSCRIBE NOW
LOG IN
Google Earth's New Voyager Tool Aims to Feed Our Wanderlust
F.C.C. Leader Seeks Tech Companies' Views on Net Neutrality
Haven't Bought a Car in a While? It's a New World Out There

https://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html

- Can be followed up with any sort of machine learning you want (e.g., OLS regression and Pearson’s correlation, naïve Bayes classifiers)

7

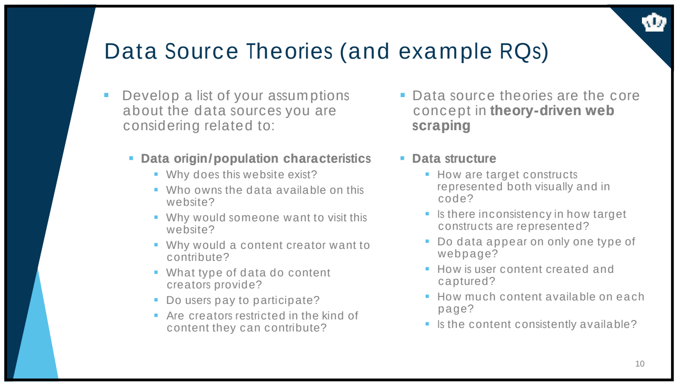
Who does this generalize to?

- That depends.
- **Landers, R. N.** & Behrend, T. S. (2015). An inconvenient truth: Arbitrary distinctions between organizations, Mechanical Turk, and other convenience samples. *Industrial and Organizational Psychology, 8,* 142-164.
  - Essentially all samples in I/O psychology are convenience samples, whether academic or practitioner research.
  - The primary questions we need to ask of any convenience sample in relation to generalizability are:
    - Omitted variables bias (endogeneity)
      - Causes of relationships/effects that come from outside our data source
    - Range restriction
      - Constraints on representativeness that comes from outside our data source

8

Endogeneity

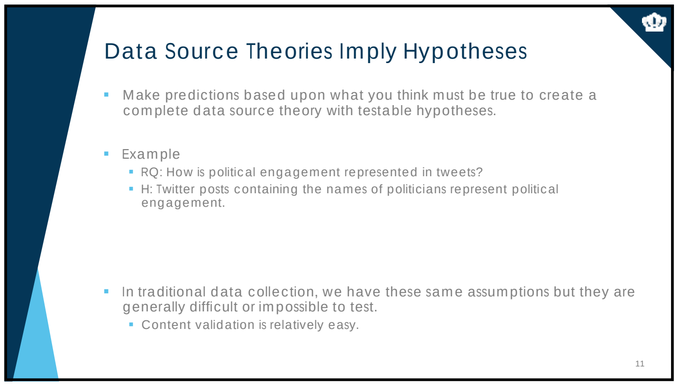
9



Data Source Theories (and example RQs)

- Develop a list of your assumptions about the data sources you are considering related to:
  - **Data origin/population characteristics**
    - Why does this website exist?
    - Who owns the data available on this website?
    - Why would someone want to visit this website?
    - Why would a content creator want to contribute?
    - What type of data do content creators provide?
    - Do users pay to participate?
    - Are creators restricted in the kind of content they can contribute?
- Data source theories are the core concept in **theory-driven web scraping**
  - **Data structure**
    - How are target constructs represented both visually and in code?
    - Is there inconsistency in how target constructs are represented?
    - Do data appear on only one type of webpage?
    - How is user content created and captured?
    - How much content available on each page?
    - Is the content consistently available?

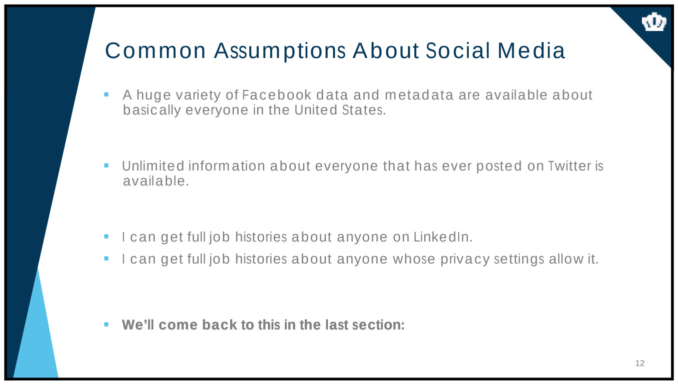
10



Data Source Theories Imply Hypotheses

- Make predictions based upon what you think must be true to create a complete data source theory with testable hypotheses.
- Example
  - RQ: How is political engagement represented in tweets?
  - H: Twitter posts containing the names of politicians represent political engagement.
- In traditional data collection, we have these same assumptions but they are generally difficult or impossible to test.
  - Content validation is relatively easy.

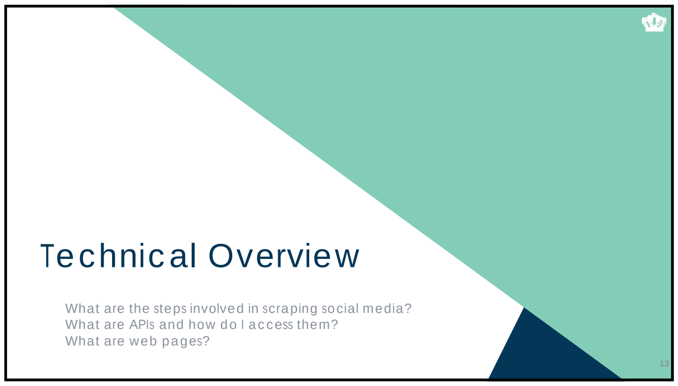
11



Common Assumptions About Social Media

- A huge variety of Facebook data and metadata are available about basically everyone in the United States.
- Unlimited information about everyone that has ever posted on Twitter is available.
- I can get full job histories about anyone on LinkedIn.
- I can get full job histories about anyone whose privacy settings allow it.
- **We'll come back to this in the last section:**

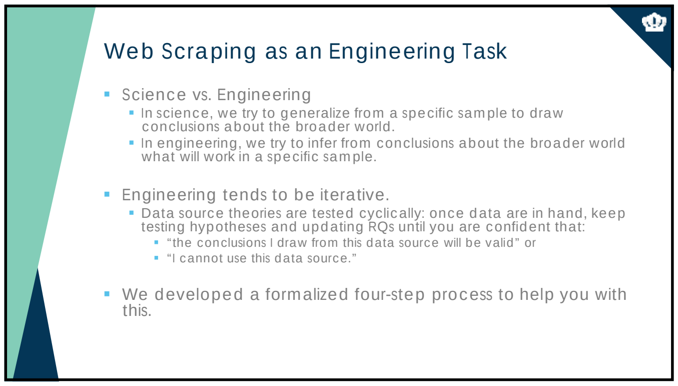
12



Technical Overview

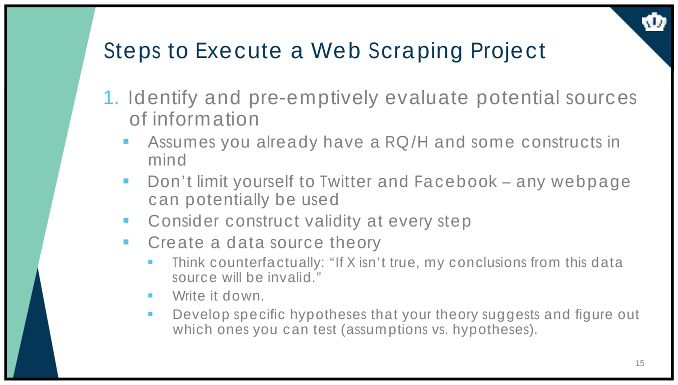
What are the steps involved in scraping social media?
What are APIs and how do I access them?
What are web pages?

13



Web Scraping as an Engineering Task

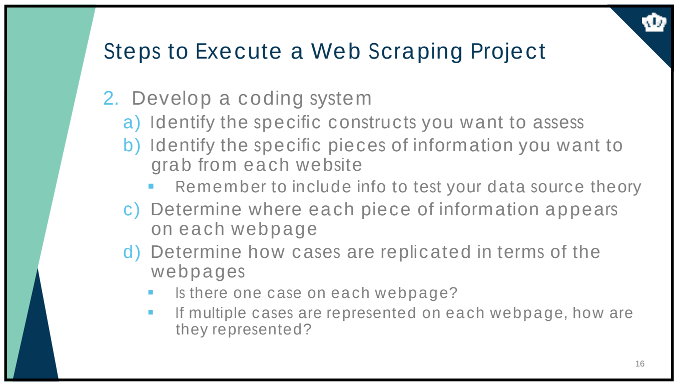
- Science vs. Engineering
  - In science, we try to generalize from a specific sample to draw conclusions about the broader world.
  - In engineering, we try to infer from conclusions about the broader world what will work in a specific sample.
- Engineering tends to be iterative.
  - Data source theories are tested cyclically: once data are in hand, keep testing hypotheses and updating RQs until you are confident that:
    - “the conclusions I draw from this data source will be valid” or
    - “I cannot use this data source.”
- We developed a formalized four-step process to help you with this.



Steps to Execute a Web Scraping Project

1. Identify and pre-emptively evaluate potential sources of information
   - Assumes you already have a RQ/H and some constructs in mind
   - Don’t limit yourself to Twitter and Facebook – any webpage can potentially be used
   - Consider construct validity at every step
   - Create a data source theory
     - Think counterfactually: “If X isn’t true, my conclusions from this data source will be invalid.”
     - Write it down.
     - Develop specific hypotheses that your theory suggests and figure out which ones you can test (assumptions vs. hypotheses).

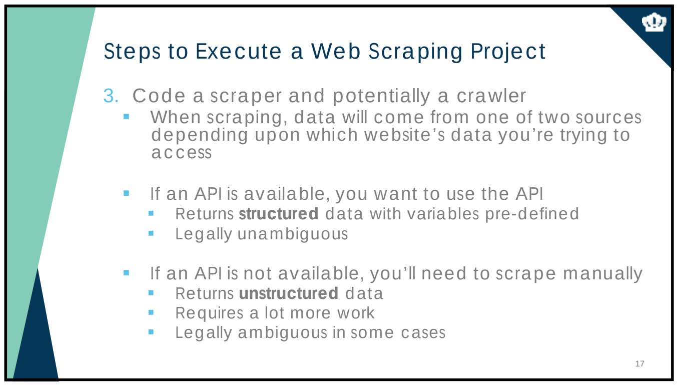
15



Steps to Execute a Web Scraping Project

2. Develop a coding system
   a) Identify the specific constructs you want to assess
   b) Identify the specific pieces of information you want to grab from each website
      - Remember to include info to test your data source theory
   c) Determine where each piece of information appears on each webpage
   d) Determine how cases are replicated in terms of the webpages
      - Is there one case on each webpage?
      - If multiple cases are represented on each webpage, how are they represented?

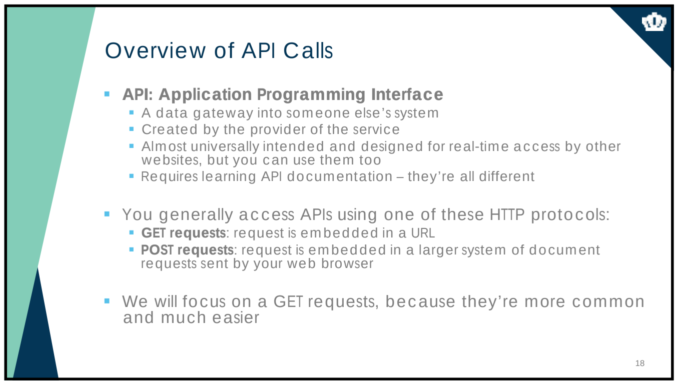
16



Steps to Execute a Web Scraping Project

3. Code a scraper and potentially a crawler
   - When scraping, data will come from one of two sources depending upon which website's data you're trying to access
   - If an API is available, you want to use the API
      - Returns **structured** data with variables pre-defined
      - Legally unambiguous
   - If an API is not available, you'll need to scrape manually
      - Returns **unstructured** data
      - Requires a lot more work
      - Legally ambiguous in some cases

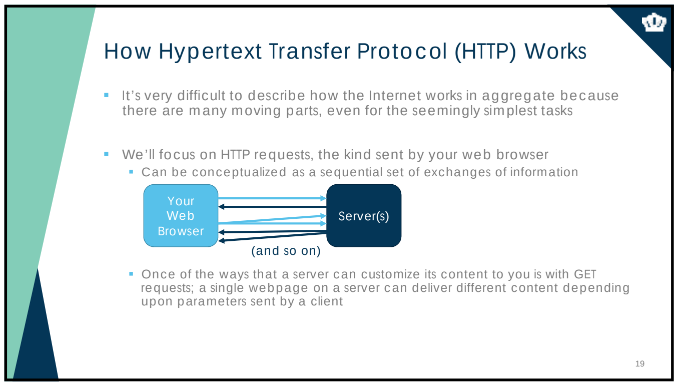
17



Overview of API Calls

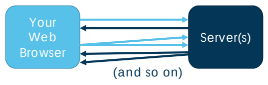
- **API: Application Programming Interface**
  - A data gateway into someone else's system
  - Created by the provider of the service
  - Almost universally intended and designed for real-time access by other websites, but you can use them too
  - Requires learning API documentation – they're all different
- You generally access APIs using one of these HTTP protocols:
  - **GET requests**: request is embedded in a URL
  - **POST requests**: request is embedded in a larger system of document requests sent by your web browser
- We will focus on a GET requests, because they're more common and much easier

18



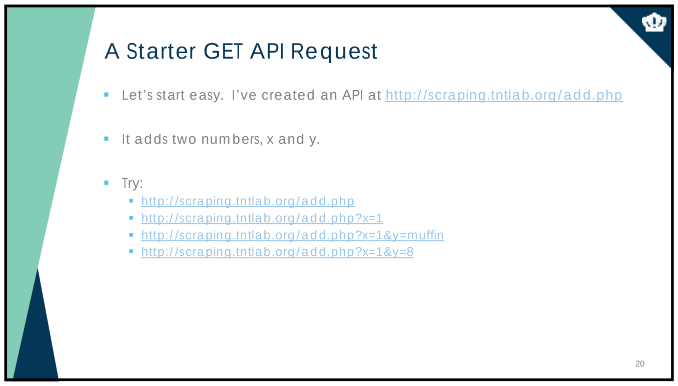
How Hypertext Transfer Protocol (HTTP) Works

- It's very difficult to describe how the Internet works in aggregate because there are many moving parts, even for the seemingly simplest tasks
- We'll focus on HTTP requests, the kind sent by your web browser
  - Can be conceptualized as a sequential set of exchanges of information



  - Once of the ways that a server can customize its content to you is with GET requests; a single webpage on a server can deliver different content depending upon parameters sent by a client

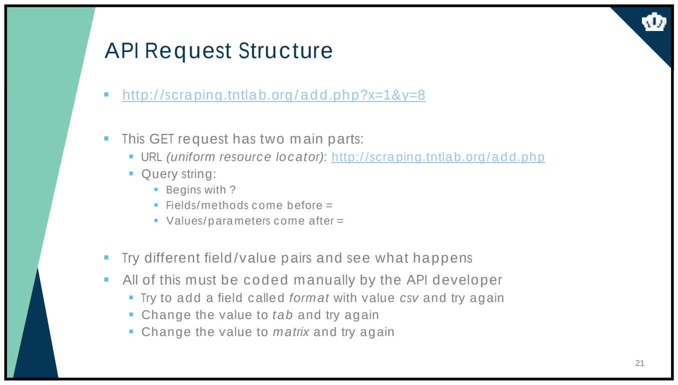
19



A Starter GET API Request

- Let's start easy. I've created an API at http://scraping.tntlab.org/add.php
- It adds two numbers, x and y.
- Try:
  - http://scraping.tntlab.org/add.php
  - http://scraping.tntlab.org/add.php?x=1
  - http://scraping.tntlab.org/add.php?x=1&y=muffin
  - http://scraping.tntlab.org/add.php?x=1&y=8

20



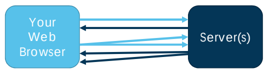
API Request Structure

- http://scraping.tntlab.org/add.php?x=1&y=8
- This GET request has two main parts:
  - URL *(uniform resource locator)*: http://scraping.tntlab.org/add.php
  - Query string:
    - Begins with ?
    - Fields/methods come before =
    - Values/parameters come after =
- Try different field/value pairs and see what happens
- All of this must be coded manually by the API developer
  - Try to add a field called *format* with value *csv* and try again
  - Change the value to *tab* and try again
  - Change the value to *matrix* and try again

21

What if there isn't an API?

- Then we need to grab data by hand and create an algorithm to provide the computer with a template for how to interpret it



- The first time you visit a webpage, your web browser sends a GET request without any fields or values
- The file that is initially returned is (usually) an HTML document: *hypertext markup language* (a specific kind of XML)
- This file is the one we will need to interpret, but without the aid of a web browser to view it
  - You've seen raw HTML yourself if you've ever clicked "View Source"

The Basic Structure of HTML: Tags

```
<html>
<head>
<title>My First Webpage</title>
</head>
<body>
<h1>My First Heading</h1>
<p>This is my first paragraph of info!<br />And a line break!</p>
<p>This is my <b>second paragraph</b> of info!</p>
</body>
</html>
```

Opening tags are just words

Self-closing tags *may* have a trailing /

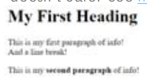
Closing tags are the same words, preceded by /

- Some tags are structural, like *html, head, title, body, h1, p*
- Some tags are inline, like *b*
- If the creator created valid HTML, nesting is always complete

The Basic Structure of HTML: Tags

```
<html>
        <head>
                <title>My First Webpage</title>
        </head>
        <body>
                <h1>My First Heading</h1>
                <p>This is my first paragraph of info!<br />And a line break!</p>
                <p>This is my <b>second paragraph</b> of info!</p>
        </body>
</html>
```

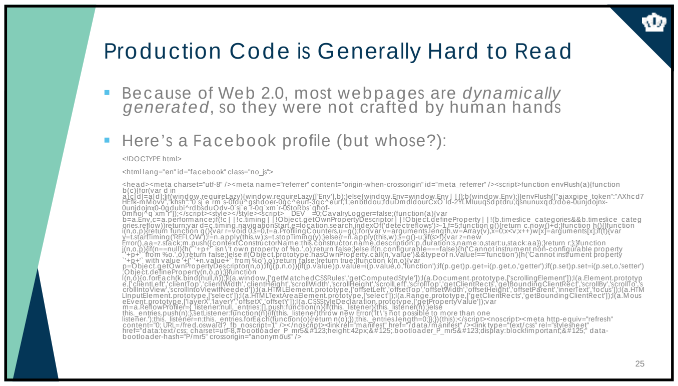
- Reorganizing HTML by its structural tags improves *readability* but your browser doesn't care. See http://scraping.tntlab.org/first.html



My First Heading

This is my first paragraph of info!
And a line break!

This is my **second paragraph** of info!



Production Code is Generally Hard to Read

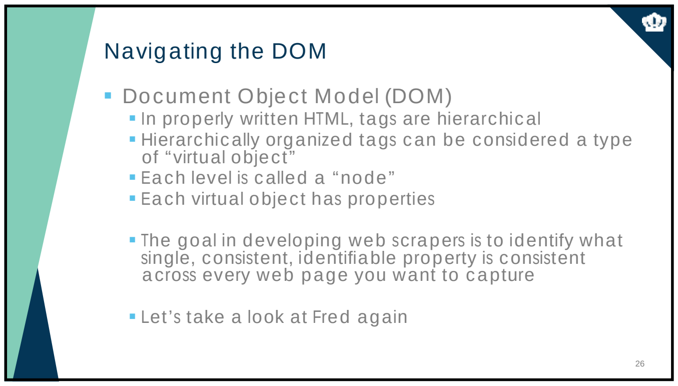
- Because of Web 2.0, most webpages are *dynamically generated*, so they were not crafted by human hands
- Here’s a Facebook profile (but whose?):

```
<!DOCTYPE html>

<html lang="en" id="facebook" class="no_js">

<head><meta charset="utf-8" /><meta name="referrer" content="origin-when-crossorigin" id="meta_referrer" /><script>function envFlush(a){function
b(c){for(var d in
a)c[d]=a[d];}if(window.requireLazy){window.requireLazy(['Env'],b);}else{window.Env=window.Env||{};b(window.Env);}}envFlush({"ajaxpipe_token":"AXhcd7
HEfk-mMovv","khsh":"0`sj`e`rm`s-0fdu^gshdoer-0gc^eurf-3gc^eurf;1;enbtldou;fduDmdldourCxO`ld-2YLMluuqSdptdru;qsnunuxqd;rdoe-0unjdojnx-
0unjdojnx0-0gdubi^rdbsduOdv-0`sj`e`r-0q`xm`r-0StoRbs`qhof-
0mhoj^q`xm`r"});</script><style></style><script>__DEV__=0;CavalryLogger=false;(function(a){var
b=a.Env,c=a.performance;if(!c||!c.timing||!Object.getOwnPropertyDescriptor||!Object.defineProperty||!(b.timeslice_categories&&b.timeslice_categ
ories.reflow))return;var d=c.timing.navigationStart,e=location.search.indexOf('detectreflows')>-1,f=5;function g(){return c.now()+d;}function h(){}function
i(n,o,p){return function q(){var r=void 0,s=0,t=a.ProfilingCounters,u=g();for(var v=arguments.length,w=Array(v),x=0;x<v;x++)w[x]=arguments[x];if(t){var
y=t.startTiming('REFLOW');r=n.apply(this,w);s=t.stopTiming(y);}else{r=n.apply(this,w);s=g()-u;}if(s>f){var z=new
Error(),aa=z.stack;m.push({contextConstructorName:this.constructor.name,description:p,duration:s,name:o,start:u,stack:aa});}return r;};}function
j(n,o,p){if(n==null){h('`'+p+'` isn\'t own property of %o.',o);return false;}else if(n.configurable===false){h('Cannot instrument non-configurable property
`'+p+'` from %o.',o);return false;}else if(Object.prototype.hasOwnProperty.call(n,'value')&&typeof n.value!=='function'){h('Cannot instrument property
`'+p+'` with value '+('`'+n.value+'` from %o'),o);return false;}return true;}function k(n,o){var
p=Object.getOwnPropertyDescriptor(n,o);if(j(p,n,o)){if(p.value)p.value=i(p.value,o,'function');if(p.get)p.get=i(p.get,o,'getter');if(p.set)p.set=i(p.set,o,'setter')
;Object.defineProperty(n,o,p);}}function
l(n,o){o.forEach(k.bind(null,n));}l(a.window,['getMatchedCSSRules','getComputedStyle']);l(a.Document.prototype,['scrollingElement']);l(a.Element.prototyp
e,['clientLeft','clientTop','clientWidth','clientHeight','scrollWidth','scrollHeight','scrollLeft','scrollTop','getClientRects','getBoundingClientRect','scrollBy','scrollTo','s
crollIntoView','scrollIntoViewIfNeeded']);l(a.HTMLElement.prototype,['offsetLeft','offsetTop','offsetWidth','offsetHeight','offsetParent','innerText','focus']);l(a.HTM
LInputElement.prototype,['select']);l(a.HTMLTextAreaElement.prototype,['select']);l(a.Range.prototype,['getClientRects','getBoundingClientRect']);l(a.Mous
eEvent.prototype,['layerX','layerY','offsetX','offsetY']);l(a.CSSStyleDeclaration.prototype,['getPropertyValue']);var
m=a.ReflowProfiler={_listener:null,_entries:[],push:function(n){if(this._listener){this._listener(n);}else
this._entries.push(n);},setListener:function(n){if(this._listener)throw new Error('It\'s not possible to more than one
listener.');this._listener=n;this._entries.forEach(function(o){return n(o);});this._entries.length=0;}};})(this);</script><noscript><meta http-equiv="refresh"
content="0; URL=/fred.oswald?_fb_noscript=1" /></noscript><link rel="manifest" href="/data/manifest" /><link type="text/css" rel="stylesheet"
href="data:text/css; charset=utf-8,#bootloader_P_mr5&#123;height:42px;&#125;.bootloader_P_mr5&#123;display:block!important;&#125;" data-
bootloader-hash="P/mr5" crossorigin="anonymous" />
```

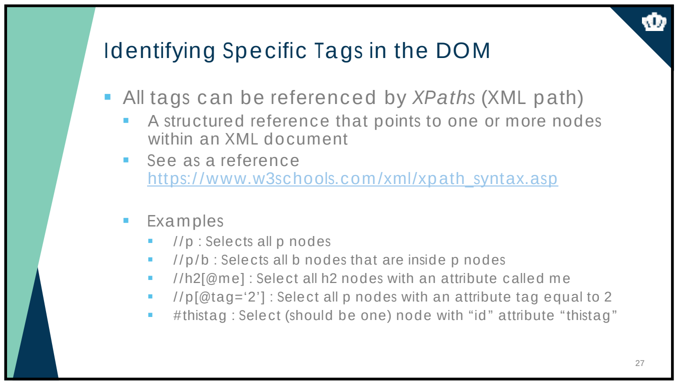
25



Navigating the DOM

- Document Object Model (DOM)
  - In properly written HTML, tags are hierarchical
  - Hierarchically organized tags can be considered a type of “virtual object”
  - Each level is called a “node”
  - Each virtual object has properties
  - The goal in developing web scrapers is to identify what single, consistent, identifiable property is consistent across every web page you want to capture
  - Let’s take a look at Fred again

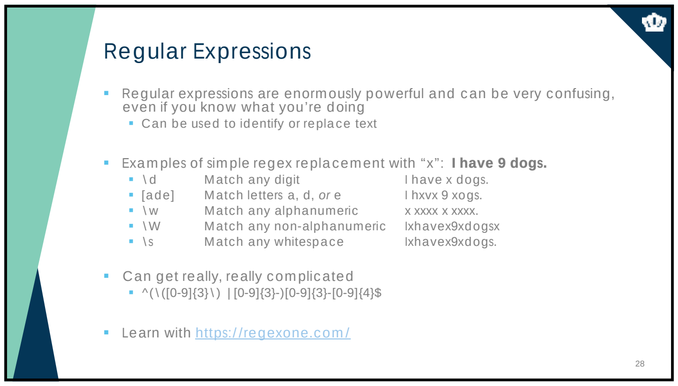
26



Identifying Specific Tags in the DOM

- All tags can be referenced by *XPaths* (XML path)
  - A structured reference that points to one or more nodes within an XML document
  - See as a reference https://www.w3schools.com/xml/xpath_syntax.asp
  - Examples
    - //p : Selects all p nodes
    - //p/b : Selects all b nodes that are inside p nodes
    - //h2[@me] : Select all h2 nodes with an attribute called me
    - //p[@tag=‘2’] : Select all p nodes with an attribute tag equal to 2
    - #thistag : Select (should be one) node with “id” attribute “thistag”

27



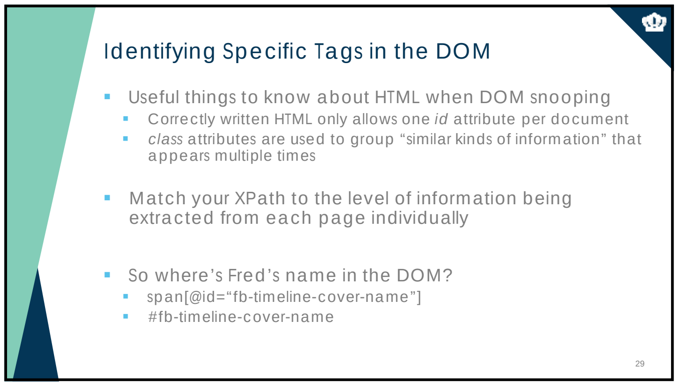
Regular Expressions

- Regular expressions are enormously powerful and can be very confusing, even if you know what you're doing
  - Can be used to identify or replace text
- Examples of simple regex replacement with "x": **I have 9 dogs.**

\d	Match any digit	I have x dogs.
[ade]	Match letters a, d, *or* e	I hxvx 9 xogs.
\w	Match any alphanumeric	x xxxx x xxxx.
\W	Match any non-alphanumeric	Ixhavex9xdogsx
\s	Match any whitespace	Ixhavex9xdogs.

- Can get really, really complicated
  - ^(\([0-9]{3}\) | [0-9]{3}-)[0-9]{3}-[0-9]{4}$
- Learn with https://regexone.com/

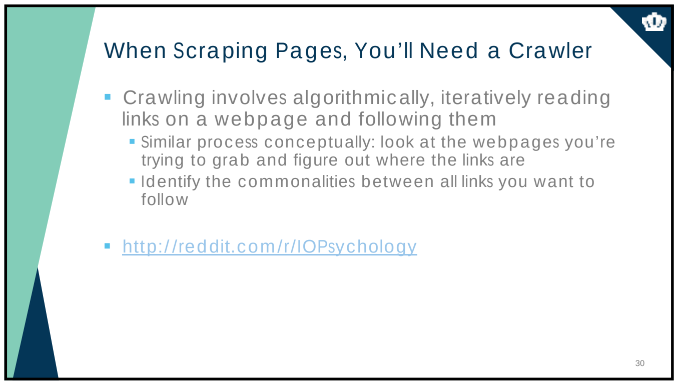
28



Identifying Specific Tags in the DOM

- Useful things to know about HTML when DOM snooping
  - Correctly written HTML only allows one *id* attribute per document
  - *class* attributes are used to group "similar kinds of information" that appears multiple times
- Match your XPath to the level of information being extracted from each page individually
- So where's Fred's name in the DOM?
  - span[@id="fb-timeline-cover-name"]
  - #fb-timeline-cover-name

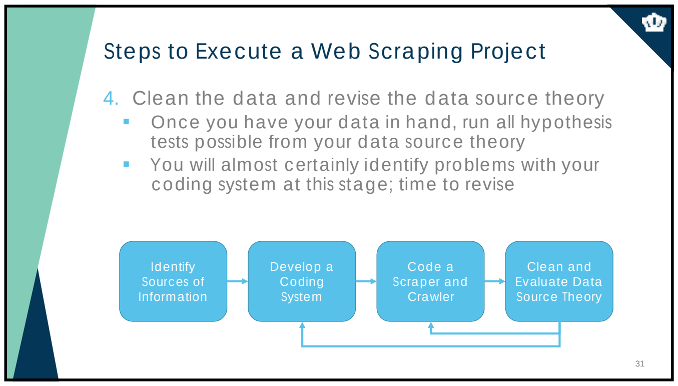
29



When Scraping Pages, You'll Need a Crawler

- Crawling involves algorithmically, iteratively reading links on a webpage and following them
  - Similar process conceptually: look at the webpages you're trying to grab and figure out where the links are
  - Identify the commonalities between all links you want to follow
- http://reddit.com/r/IOPsychology

30

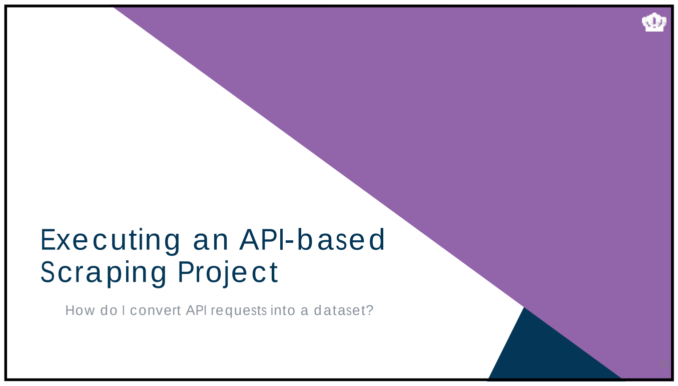


Steps to Execute a Web Scraping Project

4. Clean the data and revise the data source theory
   - Once you have your data in hand, run all hypothesis tests possible from your data source theory
   - You will almost certainly identify problems with your coding system at this stage; time to revise

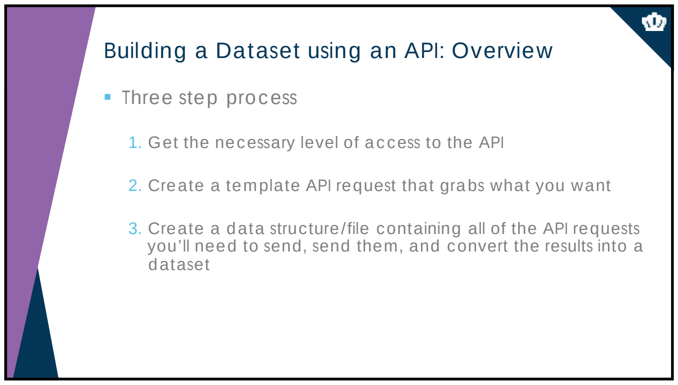
$$\begin{tikzcd}
\text{Identify Sources of Information} \arrow[r] & \text{Develop a Coding System} \arrow[r] & \text{Code a Scraper and Crawler} \arrow[r] & \text{Clean and Evaluate Data Source Theory} \arrow[ll, bend left] \arrow[l, bend left]
\end{tikzcd}$$

31



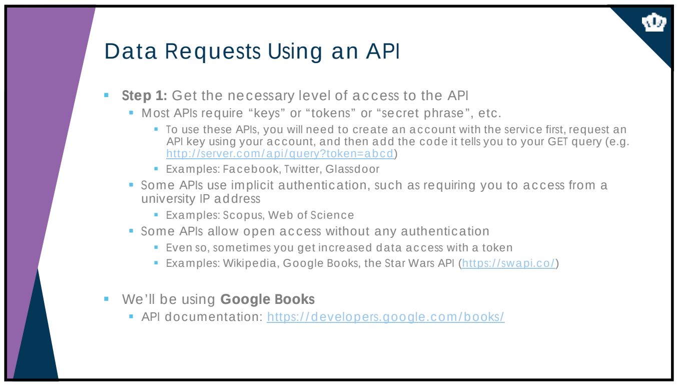
Executing an API-based Scraping Project

How do I convert API requests into a dataset?



Building a Dataset using an API: Overview

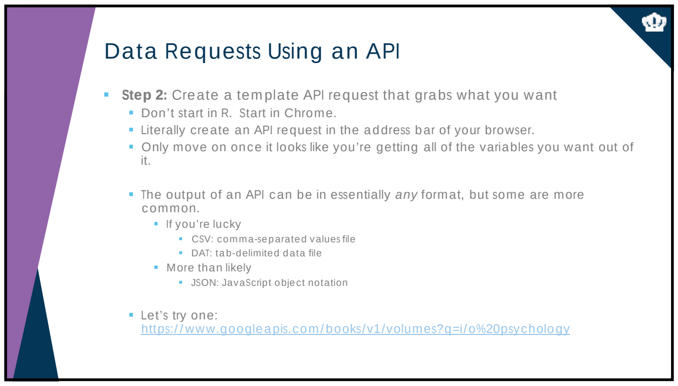
- Three step process
  1. Get the necessary level of access to the API
  2. Create a template API request that grabs what you want
  3. Create a data structure/file containing all of the API requests you'll need to send, send them, and convert the results into a dataset



Data Requests Using an API

- **Step 1:** Get the necessary level of access to the API
  - Most APIs require “keys” or “tokens” or “secret phrase”, etc.
    - To use these APIs, you will need to create an account with the service first, request an API key using your account, and then add the code it tells you to your GET query (e.g. http://server.com/api/query?token=abcd)
    - Examples: Facebook, Twitter, Glassdoor
  - Some APIs use implicit authentication, such as requiring you to access from a university IP address
    - Examples: Scopus, Web of Science
  - Some APIs allow open access without any authentication
    - Even so, sometimes you get increased data access with a token
    - Examples: Wikipedia, Google Books, the Star Wars API (https://swapi.co/)

- We’ll be using **Google Books**
  - API documentation: https://developers.google.com/books/

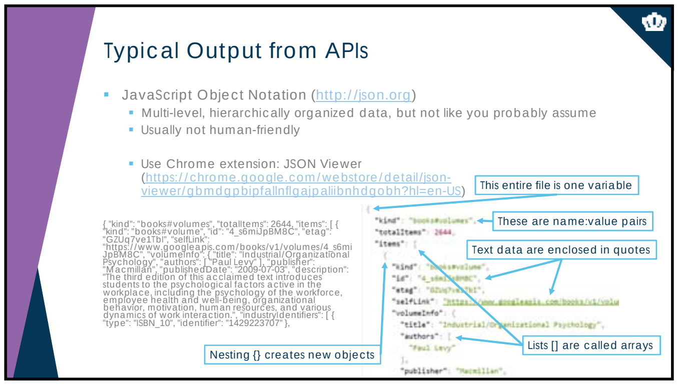


Data Requests Using an API

- **Step 2:** Create a template API request that grabs what you want
  - Don’t start in R. Start in Chrome.
  - Literally create an API request in the address bar of your browser.
  - Only move on once it looks like you’re getting all of the variables you want out of it.

  - The output of an API can be in essentially *any* format, but some are more common.
    - If you’re lucky
      - CSV: comma-separated values file
      - DAT: tab-delimited data file
    - More than likely
      - JSON: JavaScript object notation

  - Let’s try one: https://www.googleapis.com/books/v1/volumes?q=i/o%20psychology

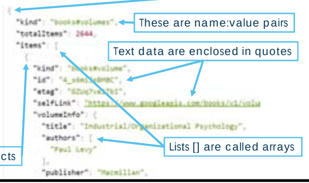


Typical Output from APIs

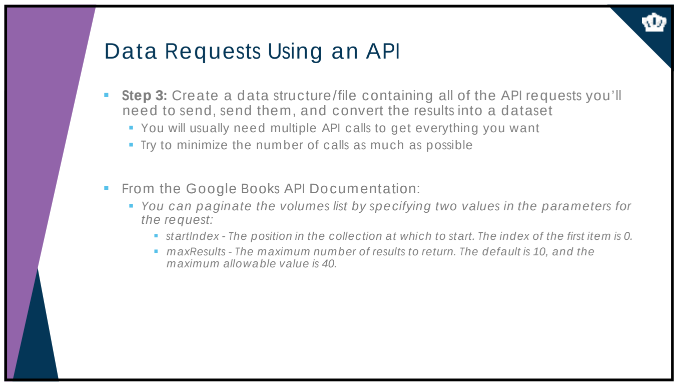
- JavaScript Object Notation (http://json.org)
  - Multi-level, hierarchically organized data, but not like you probably assume
  - Usually not human-friendly

  - Use Chrome extension: JSON Viewer (https://chrome.google.com/webstore/detail/json-viewer/gbmdgpbipfallnflgajpaliibnhdgobh?hl=en-US)

```
{ "kind": "books#volumes", "totalItems": 2644, "items": [ { "kind": "books#volume", "id": "4_s6miJpBM8C", "etag": "GZUq7ve1TbI", "selfLink": "https://www.googleapis.com/books/v1/volumes/4_s6miJpBM8C", "volumeInfo": { "title": "Industrial/Organizational Psychology", "authors": [ "Paul Levy" ], "publisher": "Macmillan", "publishedDate": "2009-07-03", "description": "The third edition of this acclaimed text introduces students to the psychological factors active in the workplace, including the psychology of the workforce, employee health and well-being, organizational behavior, motivation, human resources, and various dynamics of work interaction.", "industryIdentifiers": [ { "type": "ISBN_10", "identifier": "1429223707" },
```

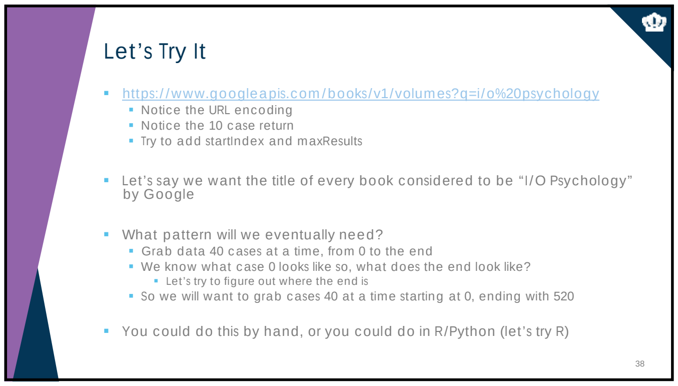

This entire file is one variable
These are name:value pairs
Text data are enclosed in quotes
Lists [] are called arrays
Nesting {} creates new objects

```
{
  "kind": "books#volumes",
  "totalItems": 2644,
  "items": [
    {
      "kind": "books#volume",
      "id": "4_s6miJpBM8C",
      "etag": "GZUq7ve1TbI",
      "selfLink": "https://www.googleapis.com/books/v1/volu
      "volumeInfo": {
        "title": "Industrial/Organizational Psychology",
        "authors": [
          "Paul Levy"
        ],
        "publisher": "Macmillan",
```



Data Requests Using an API

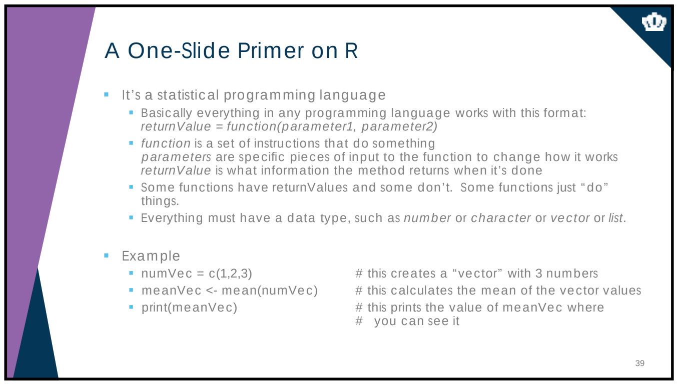
- **Step 3:** Create a data structure/file containing all of the API requests you'll need to send, send them, and convert the results into a dataset
  - You will usually need multiple API calls to get everything you want
  - Try to minimize the number of calls as much as possible
- From the Google Books API Documentation:
  - *You can paginate the volumes list by specifying two values in the parameters for the request:*
    - *startIndex - The position in the collection at which to start. The index of the first item is 0.*
    - *maxResults - The maximum number of results to return. The default is 10, and the maximum allowable value is 40.*



Let's Try It

- https://www.googleapis.com/books/v1/volumes?q=i/o%20psychology
  - Notice the URL encoding
  - Notice the 10 case return
  - Try to add startIndex and maxResults
- Let's say we want the title of every book considered to be "I/O Psychology" by Google
- What pattern will we eventually need?
  - Grab data 40 cases at a time, from 0 to the end
  - We know what case 0 looks like so, what does the end look like?
    - Let's try to figure out where the end is
  - So we will want to grab cases 40 at a time starting at 0, ending with 520
- You could do this by hand, or you could do in R/Python (let's try R)

38

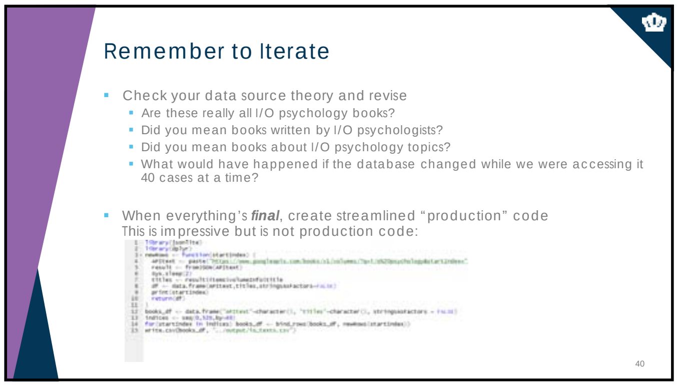


A One-Slide Primer on R

- It's a statistical programming language
  - Basically everything in any programming language works with this format: *returnValue = function(parameter1, parameter2)*
  - *function* is a set of instructions that do something
    parameters are specific pieces of input to the function to change how it works
    returnValue is what information the method returns when it's done
  - Some functions have returnValues and some don't. Some functions just "do" things.
  - Everything must have a data type, such as *number* or *character* or *vector* or *list*.
- Example

```
numVec = c(1,2,3)            # this creates a "vector" with 3 numbers
meanVec <- mean(numVec)      # this calculates the mean of the vector values
print(meanVec)               # this prints the value of meanVec where
                             #   you can see it
```

39



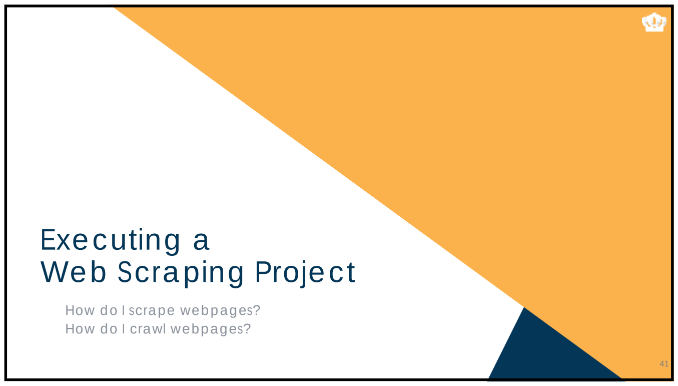
Remember to Iterate

- Check your data source theory and revise
  - Are these really all I/O psychology books?
  - Did you mean books written by I/O psychologists?
  - Did you mean books about I/O psychology topics?
  - What would have happened if the database changed while we were accessing it 40 cases at a time?
- When everything's ***final***, create streamlined "production" code

This is impressive but is not production code:

```
library(jsonlite)
library(dplyr)
newRows <- function(startIndex) {
  APItext <- paste("https://www.googleapis.com/books/v1/volumes?q=I/O%20psychology&startIndex="
  result <- fromJSON(APItext)
  Sys.sleep(2)
  titles <- result$items$volumeInfo$title
  df <- data.frame(APItext,titles,stringsAsFactors=FALSE)
  print(startIndex)
  return(df)
}
books_df <- data.frame("APItext"=character(), "titles"=character(), stringsAsFactors = FALSE)
indices <- seq(0,520,by=40)
for(startIndex in indices) books_df <- bind_rows(books_df, newRows(startIndex))
write.csv(books_df, "../output/io_texts.csv")
```

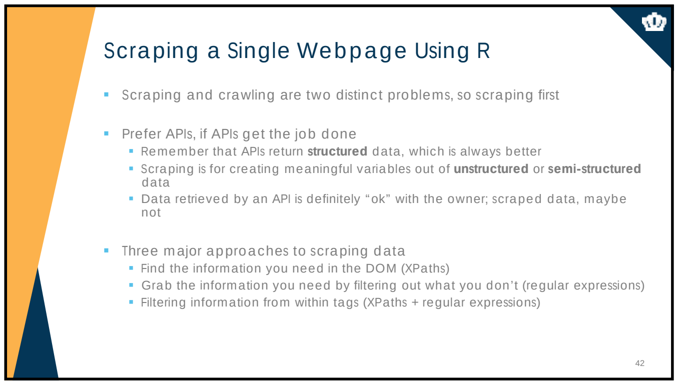
40



Executing a Web Scraping Project

How do I scrape webpages?
How do I crawl webpages?

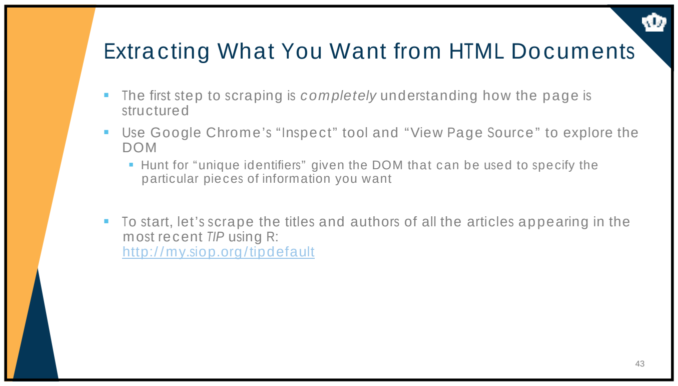
41



Scraping a Single Webpage Using R

- Scraping and crawling are two distinct problems, so scraping first
- Prefer APIs, if APIs get the job done
  - Remember that APIs return **structured** data, which is always better
  - Scraping is for creating meaningful variables out of **unstructured** or **semi-structured** data
  - Data retrieved by an API is definitely "ok" with the owner; scraped data, maybe not
- Three major approaches to scraping data
  - Find the information you need in the DOM (XPaths)
  - Grab the information you need by filtering out what you don't (regular expressions)
  - Filtering information from within tags (XPaths + regular expressions)

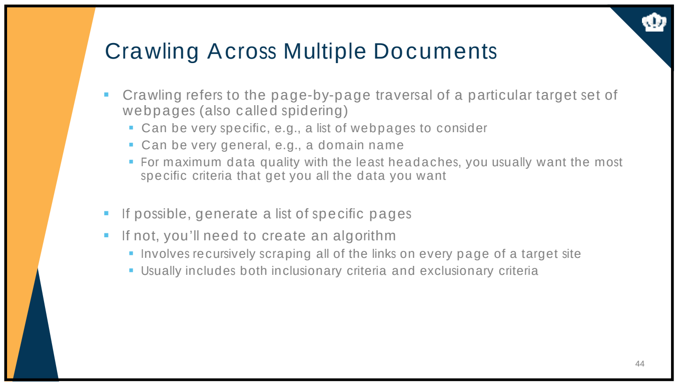
42



Extracting What You Want from HTML Documents

- The first step to scraping is *completely* understanding how the page is structured
- Use Google Chrome's "Inspect" tool and "View Page Source" to explore the DOM
  - Hunt for "unique identifiers" given the DOM that can be used to specify the particular pieces of information you want
- To start, let's scrape the titles and authors of all the articles appearing in the most recent *TIP* using R: http://my.siop.org/tipdefault

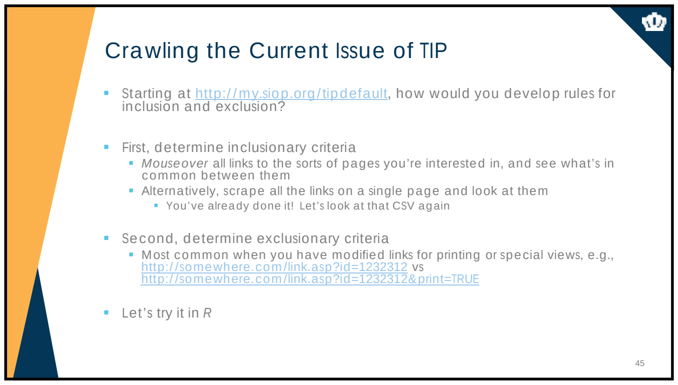
43



Crawling Across Multiple Documents

- Crawling refers to the page-by-page traversal of a particular target set of webpages (also called spidering)
  - Can be very specific, e.g., a list of webpages to consider
  - Can be very general, e.g., a domain name
  - For maximum data quality with the least headaches, you usually want the most specific criteria that get you all the data you want
- If possible, generate a list of specific pages
- If not, you'll need to create an algorithm
  - Involves recursively scraping all of the links on every page of a target site
  - Usually includes both inclusionary criteria and exclusionary criteria

44



Crawling the Current Issue of TIP

- Starting at http://my.siop.org/tipdefault, how would you develop rules for inclusion and exclusion?
- First, determine inclusionary criteria
  - *Mouseover* all links to the sorts of pages you're interested in, and see what's in common between them
  - Alternatively, scrape all the links on a single page and look at them
    - You've already done it! Let's look at that CSV again
- Second, determine exclusionary criteria
  - Most common when you have modified links for printing or special views, e.g., http://somewhere.com/link.asp?id=1232312 vs http://somewhere.com/link.asp?id=1232312&print=TRUE
- Let's try it in *R*

45

Crawling then Scraping

- This was the easiest type of crawling: there is a single link of URLs that you can scrape individually
- Recursive crawling is the hardest: any webpage you crawl may contain *new* links that in turn need to be crawled. To do this, you'll need to:
  - Crawl an initial set of webpages/link
  - Within each of those webpages, scrape all embedded links
  - Process links according to inclusionary/exclusionary rules
  - Create a new list of "scrape next" links
  - Return to step 1 with new list

46

This is Why You Want an API

- Crawling/scraping is more complicated than API requests because you are restricted by:
  - Often poorly written webpages that are non-compliant with the HTML standards (to see if you're crazy, check https://validator.w3.org)
  - Nonsensical pagination and naming conventions
  - Dynamic webpages that don't create distinct URLs (http://www.siop.org/jobnet/default.aspx)
  - Server-side restrictions, such as crawling speed
  - Your own coding skill, attention to detail, and patience
- R is also not particularly well-suited for crawling
  - This is where I suggest you turn to the *scrapy* library in Python

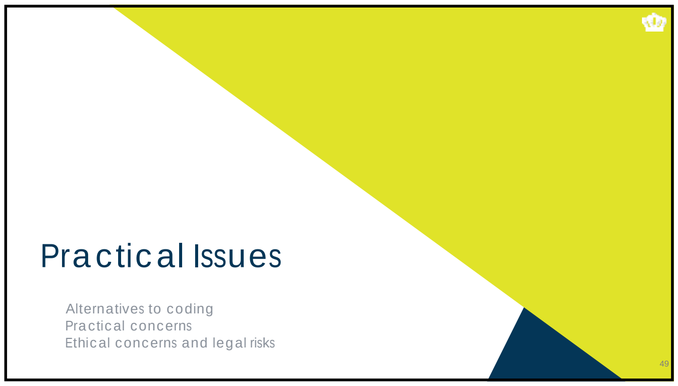
For Big-Data Scientists, 'Janitor Work' Is Key Hurdle to Insights

47

To Learn More Technical Bits

- For general information about both *R* and *Python*, I strongly recommend http://datacamp.com
- General Crawling/Scraping Frameworks
  - To learn how to use *scrapy* with Python, I recommend my tutorial: http://rlanders.net/scrapytutorial.html
  - The other big library for web crawling/scraping in Python is *Beautiful Soup:* https://www.crummy.com/software/BeautifulSoup/
- Parsing
  - To learn basic HTML and CSS: https://www.codecademy.com/learn/web
  - To learn how to use XPath: http://www.w3schools.com/xpath/
  - To learn how to use regular expressions: https://regexone.com/

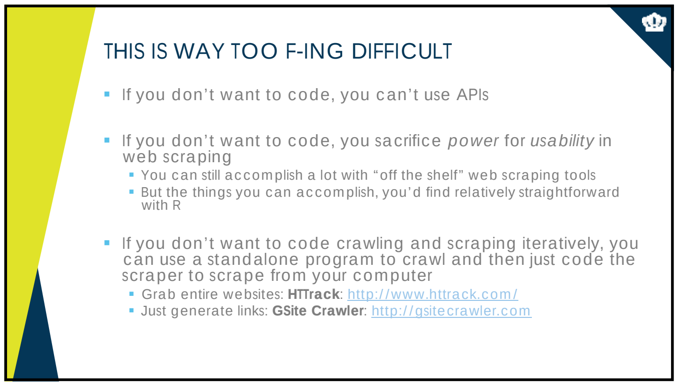
48



Practical Issues

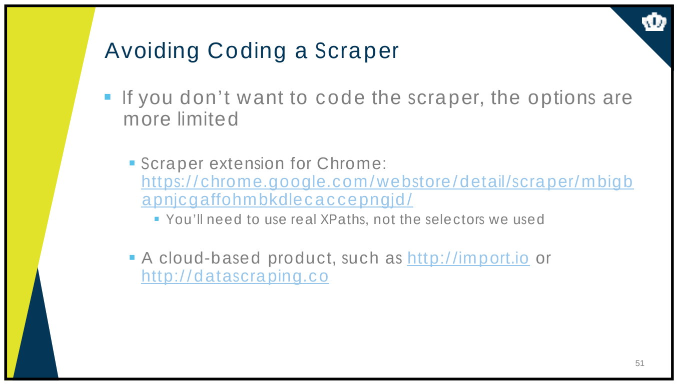
Alternatives to coding
Practical concerns
Ethical concerns and legal risks

49



THIS IS WAY TOO F-ING DIFFICULT

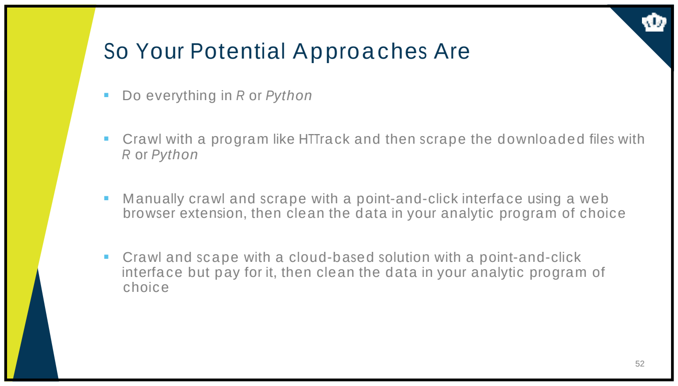
- If you don't want to code, you can't use APIs
- If you don't want to code, you sacrifice *power* for *usability* in web scraping
  - You can still accomplish a lot with "off the shelf" web scraping tools
  - But the things you can accomplish, you'd find relatively straightforward with R
- If you don't want to code crawling and scraping iteratively, you can use a standalone program to crawl and then just code the scraper to scrape from your computer
  - Grab entire websites: **HTTrack**: http://www.httrack.com/
  - Just generate links: **GSite Crawler**: http://gsitecrawler.com



Avoiding Coding a Scraper

- If you don't want to code the scraper, the options are more limited
  - Scraper extension for Chrome: https://chrome.google.com/webstore/detail/scraper/mbigbapnjcgaffohmbkdlecaccepngjd/
    - You'll need to use real XPaths, not the selectors we used
  - A cloud-based product, such as http://import.io or http://datascraping.co

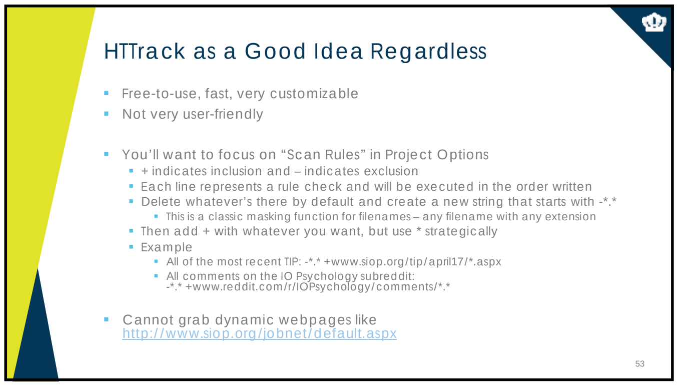
51



So Your Potential Approaches Are

- Do everything in *R* or *Python*
- Crawl with a program like HTTrack and then scrape the downloaded files with *R* or *Python*
- Manually crawl and scrape with a point-and-click interface using a web browser extension, then clean the data in your analytic program of choice
- Crawl and scape with a cloud-based solution with a point-and-click interface but pay for it, then clean the data in your analytic program of choice

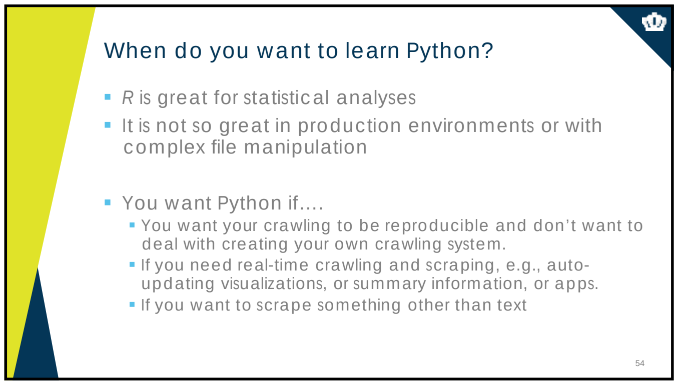
52



HTTrack as a Good Idea Regardless

- Free-to-use, fast, very customizable
- Not very user-friendly
- You'll want to focus on "Scan Rules" in Project Options
  - + indicates inclusion and – indicates exclusion
  - Each line represents a rule check and will be executed in the order written
  - Delete whatever's there by default and create a new string that starts with -*.*
    - This is a classic masking function for filenames – any filename with any extension
  - Then add + with whatever you want, but use * strategically
  - Example
    - All of the most recent TIP: -*.* +www.siop.org/tip/april17/*.aspx
    - All comments on the IO Psychology subreddit:
      -*.* +www.reddit.com/r/IOPsychology/comments/*.*
- Cannot grab dynamic webpages like
  http://www.siop.org/jobnet/default.aspx

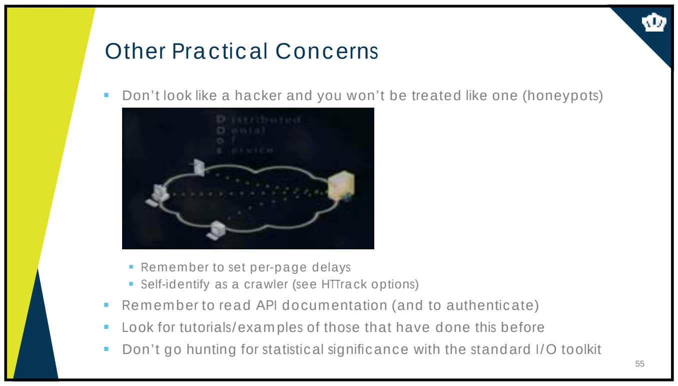
53



When do you want to learn Python?

- *R* is great for statistical analyses
- It is not so great in production environments or with complex file manipulation
- You want Python if….
  - You want your crawling to be reproducible and don't want to deal with creating your own crawling system.
  - If you need real-time crawling and scraping, e.g., auto-updating visualizations, or summary information, or apps.
  - If you want to scrape something other than text

54

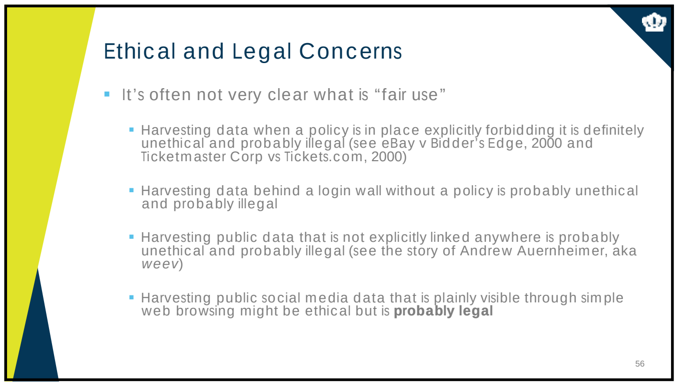


Other Practical Concerns

- Don’t look like a hacker and you won’t be treated like one (honeypots)

  - Remember to set per-page delays
  - Self-identify as a crawler (see HTTrack options)
- Remember to read API documentation (and to authenticate)
- Look for tutorials/examples of those that have done this before
- Don’t go hunting for statistical significance with the standard I/O toolkit

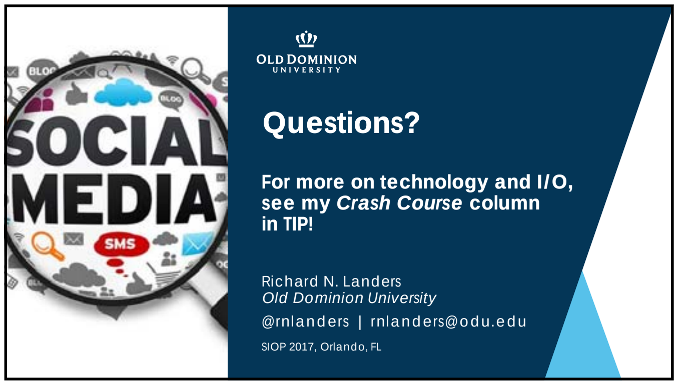
55



Ethical and Legal Concerns

- It’s often not very clear what is “fair use”
  - Harvesting data when a policy is in place explicitly forbidding it is definitely unethical and probably illegal (see eBay v Bidder’s Edge, 2000 and Ticketmaster Corp vs Tickets.com, 2000)
  - Harvesting data behind a login wall without a policy is probably unethical and probably illegal
  - Harvesting public data that is not explicitly linked anywhere is probably unethical and probably illegal (see the story of Andrew Auernheimer, aka *weev*)
  - Harvesting public social media data that is plainly visible through simple web browsing might be ethical but is **probably legal**

56



OLD DOMINION UNIVERSITY

Questions?

For more on technology and I/O, see my *Crash Course* column in TIP!

Richard N. Landers
Old Dominion University
@rnlanders | rnlanders@odu.edu

SIOP 2017, Orlando, FL