INTRODUCTION TO DATA SCIENCE

JOHN P DICKERSON

Lecture #2 – 01/31/2017

CMSC320
Tuesdays & Thursdays
3:30pm – 4:45pm

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

ANNOUNCEMENTS

Register on Piazza: piazza.com/umd/spring2017/cmsc320

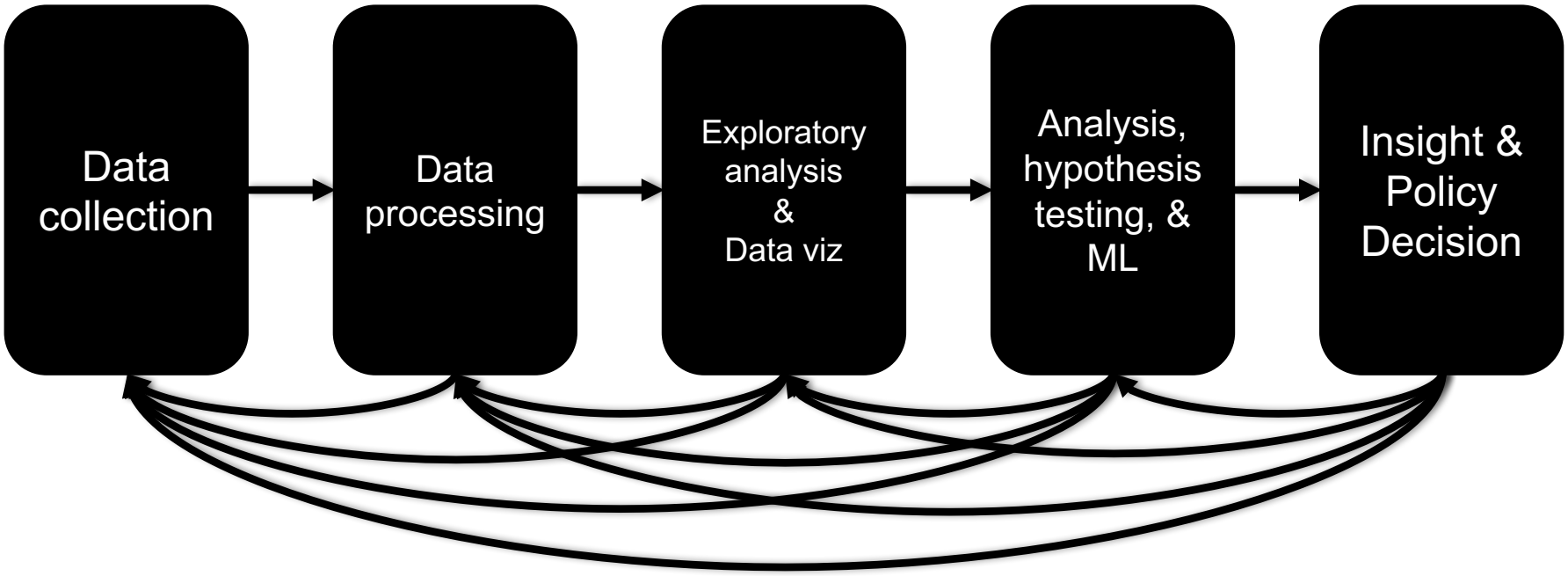
- 64 have registered already ♡
- 21 have not registered yet 💔

I will be travelling Friday night–Tuesday night:

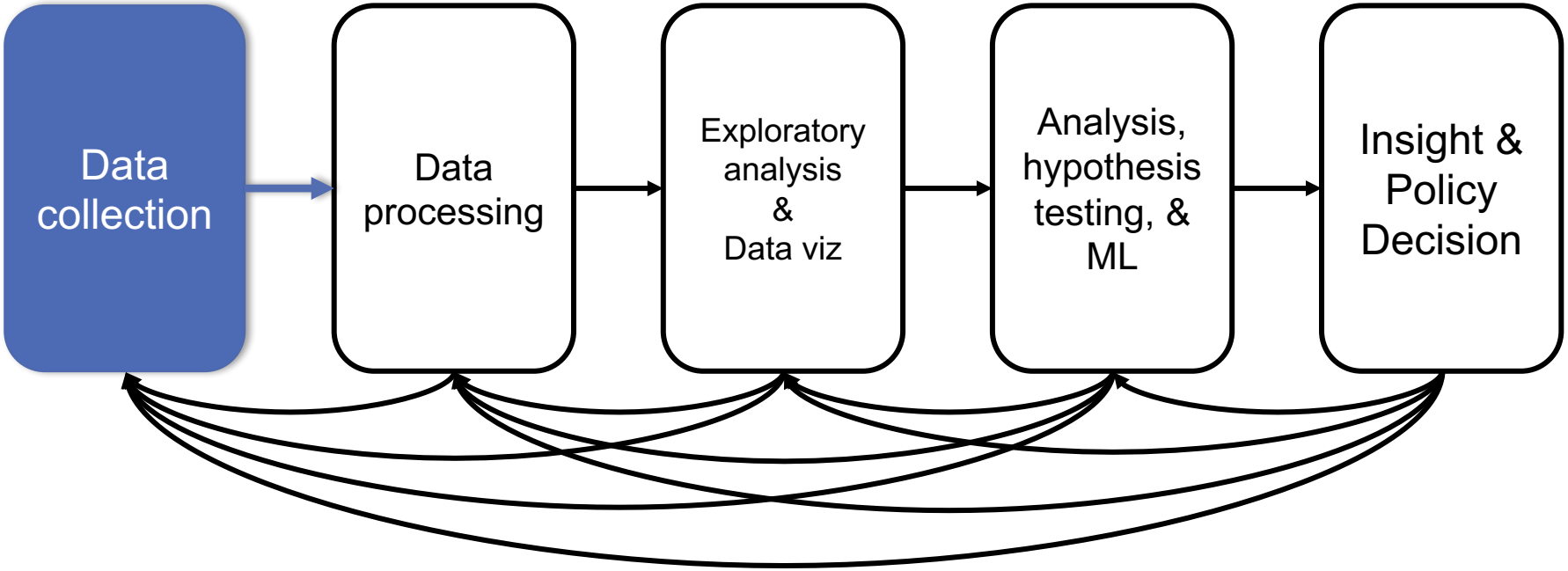
- Denis, Neil, & Anant will run a Python tutorial on Tuesday (2/7)
- I will probably still hold office hours on Friday (2/3)

?

THE DATA LIFECYCLE



TODAY'S LECTURE



BUT FIRST, SNAKES!

Python is an interpreted, dynamically-typed, high-level, garbage-collected, object-oriented-functional-imperative, and widely used scripting language.

- **Interpreted:** instructions executed without being compiled into (virtual) machine instructions*
- **Dynamically-typed:** verifies type safety at runtime
- **High-level:** abstracted away from the raw metal and kernel
- **Garbage-collected:** memory management is automated
- **OOFI:** you can do bits of OO, F, and I programming

Not the point of this class!

- Python is fast (developer time), intuitive, and used in industry!

*you can compile Python source, but it's not required

THE ZEN OF PYTHON

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules …
- … although practicality beats purity.
- Errors should never pass silently …
- … unless explicitly silenced.

Thanks: SDSMT ACM/LUG

LITERATE PROGRAMMING

Literate code contains in one document:

- the source code;
- text explanation of the code; and
- the end result of running the code.

Basic idea: present code in the order that logic and flow of human thoughts demand, not the machine-needed ordering

- Necessary for data science!
- Many choices made need textual explanation, ditto results.

Lab next Tuesday in lecture!

IP[y]: IPython Interactive Computing

jupyter

10-MINUTE PYTHON PRIMER

Define a function:

```
def my_func(x, y):
    if x > y:
        return x
    else:
        return y
```

Python is whitespace-delimited

Define a function that returns a tuple:

```
def my_func(x, y):
    return (x-1, y+2)

(a, b) = my_func(1, 2)
```

```
a = 0; b = 4
```

USEFUL BUILT-IN FUNCTIONS: COUNTING AND ITERATING

`len`: returns the number of items of an enumerable object

```
len( ['c', 'm', 's', 'c', 3, 2, 0] )
```

```
7
```

`range`: returns an iterable object

```
list( range(10) )
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

`enumerate`: returns iterable tuple (index, element) of a list

```
enumerate( ["311", "320", "330"] )
```

```
[(0, "311"), (1, "320"), (2, "330")]
```

https://docs.python.org/3/library/functions.html

USEFUL BUILT-IN FUNCTIONS: MAP AND FILTER

`map`: apply a function to a sequence or iterable

```
arr = [1, 2, 3, 4, 5]
map(lambda x: x**2, arr)
```

```
[1, 4, 9, 16, 25]
```

`filter`: returns a list of elements for which a predicate is true

```
arr = [1, 2, 3, 4, 5, 6, 7]
filter(lambda x: x % 2 == 0, arr)
```

```
[2, 4, 6]
```

We'll go over in much greater depth with pandas/numpy.

PYTHONIC PROGRAMMING

Basic iteration over an array in Java:

```
int[] arr = new int[10];
for(int idx=0; idx<arr.length; ++idx) {
    System.out.println( arr[idx] );
}
```

Direct translation into Python:

```
idx = 0
while idx < len(arr):
    print( arr[idx] ); idx += 1
```

A more “Pythonic” way of iterating:

```
for element in arr:
    print( element )
```

LIST COMPREHENSIONS

Construct sets like a mathematician!

- $P = \{ 1, 2, 4, 8, 16, \ldots, 2^{16} \}$
- $E = \{ x \mid x \text{ in } \mathbb{N} \text{ and } x \text{ is odd and } x < 1000 \}$

Construct lists like a mathematician who codes!

```
P = [ x**2 for x in range(17) ]
```

```
E = [ x for x in range(1000) if x % 2 != 0 ]
```

Very similar to `map`, but:

- **You'll see these way more than `map` in the wild**
- **Many people consider `map`/`filter` not "pythonic"**
- **They can perform differently (`map` is "lazier")**

EXCEPTIONS

Syntactically correct statement throws an exception:

- `tweepy` (Python Twitter API) returns “Rate limit exceeded”
- sqlite (a file-based database) returns `IntegrityError`

```
print('Python', python_version())

try:
   cause_a_NameError
except NameError as err:
   print(err, '-> some extra text')
```

PYTHON 2 VS 3

Python 3 is intentionally backwards incompatible

- (But not *that* incompatible)

Biggest changes that matter for us:

- `print "statement"` → `print("function")`
- `1/2 = 0` → `1/2 = 0.5` and `1//2 = 0`
- ASCII `str` default → default Unicode

Namespace ambiguity fixed:

```
i = 1
[i for i in range(5)]
print(i)   # ????????
```

TO ANY CURMUDGEONS …

If you're going to use Python 2 anyway, use the `_future_` module:

- Python 3 introduces features that will throw runtime errors in Python 2 (e.g., `with` statements)
- `_future_` module incrementally brings 3 functionality into 2
- https://docs.python.org/2/library/__future__.html

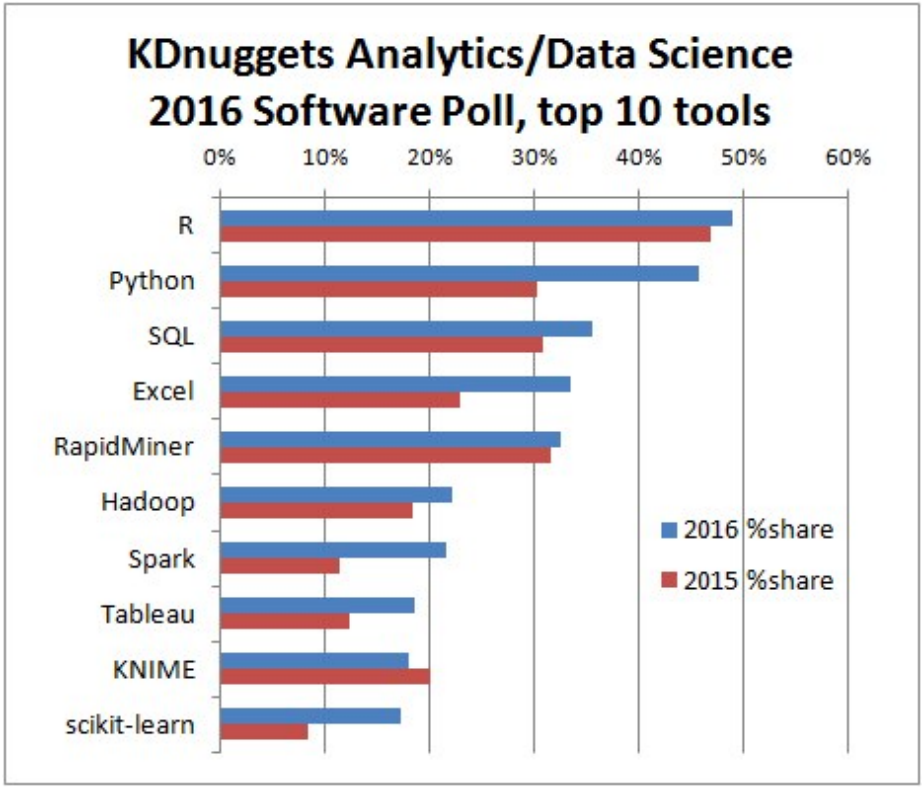
```
from _future_ import division
from _future_ import print_function
from _future_ import please_just_use_python_3
```

PYTHON VS R (FOR DATA SCIENTISTS)

There is no right answer here!

- **Python is a "full" programming language – easier to integrate with systems in the field**
- **R has a more mature set of pure stats libraries …**
- **… but Python is catching up quickly …**
- **… and is already ahead specifically for ML.**

You will see Python more in the tech industry.



EXTRA RESOURCES

Plenty of tutorials on the web:

- https://www.learnpython.org/

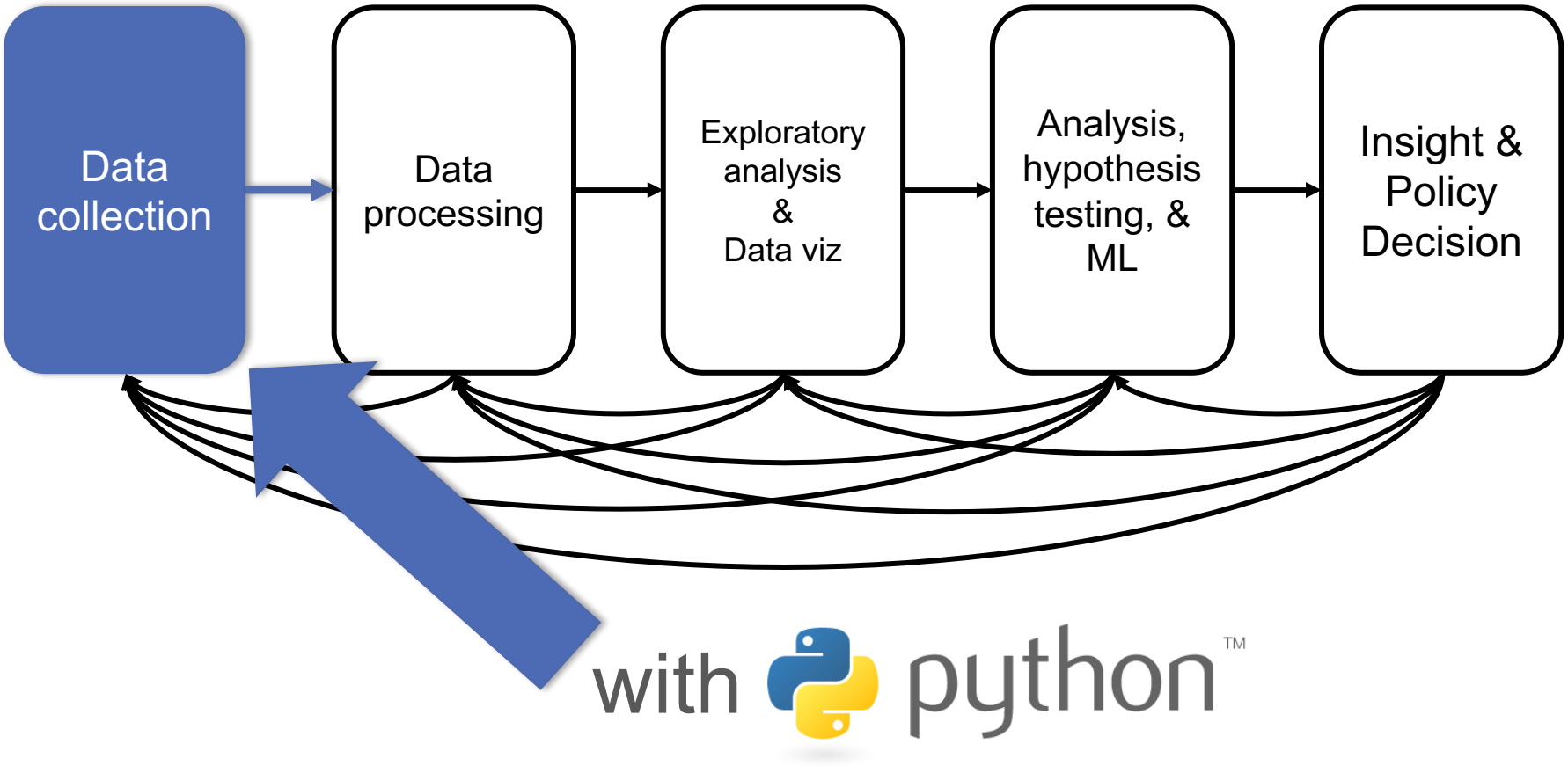
Lecture next Tuesday (2/7) will be an interactive, in-class Jupyter tutorial:

- Bring a laptop and follow along!

Come hang out at office hours (or chat with me privately)

- All office hours are now posted on the website/Piazza

TODAY'S LECTURE



Thanks: Zico Kolter's 15-388

GOTTA CATCH ‘EM ALL

Five ways to get data:

- **Direct download and load from local storage**
- **Generate locally via downloaded code (e.g., simulation)**
- **Query data from a database (covered 2/16)**
- **Query an API from the intra/internet**
- **Scrape data from a webpage**

} Covered today and possibly Thursday

WHEREFORE ART THOU, API?

A web-based Application Programming Interface (API) like we'll be using in this class is a contract between a server and a user stating:

> **"If you send me a specific request, I will return some information in a structured and documented format."**

(More generally, APIs can also perform actions, may not be web-based, be a set of protocols for communicating between processes, between an application and an OS, etc.)

“SEND ME A SPECIFIC REQUEST”

Most web API queries we’ll be doing will use HTTP requests:

- `conda install –c anaconda requests=2.12.4`

```
r = requests.get(   'https://api.github.com/user',
                    auth=('user', 'pass')   )
```

```
r.status_code
```

```
200
```

```
r.headers[‘content-type’]
```

```
‘application/json; charset=utf8’
```

```
r.json()
```

```
{u'private_gists': 419, u'total_private_repos': 77, ...}
```

http://docs.python-requests.org/en/master/

HTTP REQUESTS

https://www.google.com/**?q=cmsc320&tbs=qdr:m**

??????????

HTTP GET Request:

GET /?q=cmsc320&tbs=qdr:m HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:10.0.1) Gecko/20100101 Firefox/10.0.1

```
params = { “q”: “cmsc320”, “tbs”: “qdr:m” }
r = requests.get(    “https://www.google.com”,
                     params = params )
```

*be careful with https:// calls; `requests` will not verify SSL by default

RESTFUL APIS

This class will just query web APIs, but full web APIs typically allow more.

Representational State Transfer (RESTful) APIs:

- GET: perform query, return data
- POST: create a new entry or object
- PUT: update an existing entry or object
- DELETE: delete an existing entry or object

Can be more intricate, but verbs (“put”) align with actions

QUERYING A RESTFUL API

Stateless: with every request, you send along a token/authentication of who you are

```
token = "super_secret_token"
r = requests.get("https://github.com/user",
                 params={"access_token": token})
print( r.content )
```

```
{"login":"JohnDickerson","id":472985,"avatar_url":"ht…
```

GitHub is more than a GETHub:

- PUT/POST/DELETE can edit your repositories, etc.
- Try it out: https://github.com/settings/tokens/new

AUTHENTICATION AND OAUTH

Old and busted:

```
r = requests.get("https://api.github.com/user",
                 auth=("JohnDickerson", "ILoveKittens"))
```

New hotness:

- What if I wanted to grant an app access to, e.g., my Facebook account without giving that app my password?
- OAuth: grants access tokens that give (possibly incomplete) access to a user or app without exposing a password

"… I WILL RETURN INFORMATION IN A STRUCTURED FORMAT."

So we've queried a server using a well-formed GET request via the `requests` Python module. What comes back?

General structured data:

- Comma-Separated Value (CSV) files & strings
- Javascript Object Notation (JSON) files & strings
- HTML, XHTML, XML files & strings

Domain-specific structured data:

- Shapefiles: geospatial vector data (OpenStreetMap)
- RVT files: architectural planning (Autodesk Revit)
- You can make up your own! Always document it.

CSV FILES IN PYTHON

Any CSV reader worth anything can parse files with any delimiter, not just a comma (e.g., “TSV” for tab-separated)

1,26-Jan,Introduction,—,"pdf, pptx",Dickerson,
2,31-Jan,Scraping Data with Python,Anaconda's Test Drive.,,Dickerson,
3,2-Feb,"Vectors, Matrices, and Dataframes",Introduction to pandas.,,Dickerson,
4,7-Feb,Jupyter notebook lab,,,"Denis, Anant, & Neil",
5,9-Feb,Best Practices for Data Science Projects,,,Dickerson,

Don’t write your own CSV or JSON parser

```
import csv
with open(“schedule.csv”, ”rb”) as f:
    reader = csv.reader(f, delimiter=“,”, quotechar=’”’)
    for row in reader:
        print(row)
```

(We’ll use `pandas` to do this much more easily and efficiently)

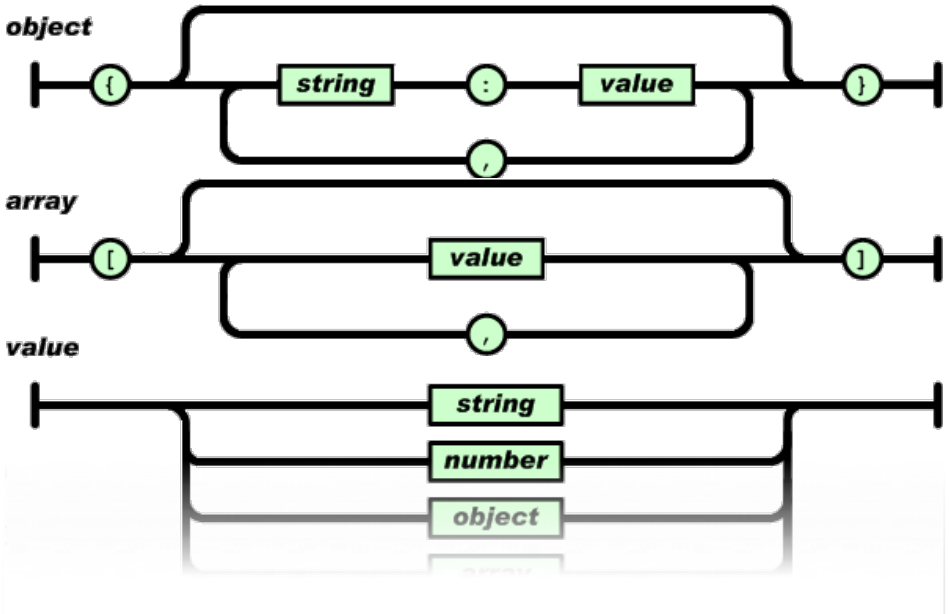
JSON FILES & STRINGS

JSON is a method for serializing objects:

- Convert an object into a string (done in Java in 131/132?)
- Deserialization converts a string back to an object

Easy for humans to read (and sanity check, edit)

Defined by three universal data structures



Python dictionary, Java Map, hash table, etc …

Python list, Java array, vector, etc …

Python string, float, int, boolean, JSON object, JSON array, …

Images from: http://www.json.org/

JSON IN PYTHON

Some built-in types: `"Strings", 1.0, True, False, None`

Lists: `["Goodbye", "Cruel", "World"]`

Dictionaries: `{"hello": "bonjour", "goodbye", "au revoir"}`

Dictionaries within lists within dictionaries within lists:

```
[1, 2, {"Help":[
                "I'm", {"trapped": "in"},
                "CMSC320"
                ]}]
```

JSON FROM TWITTER

```
GET https://api.twitter.com/1.1/friends/list.json?cursor=-
1&screen_name=twitterapi&skip_status=true&include_user_entitie
s=false
```


```
{
        "previous_cursor": 0,
        "previous_cursor_str": "0",
        "next_cursor": 1333504313713126852,
        "users": [{
                "profile_sidebar_fill_color": "252429",
                "profile_sidebar_border_color": "181A1E",
                "profile_background_tile": false,
                "name": "Sylvain Carle",
                "profile_image_url":
 "http://a0.twimg.com/profile_images/2838630046/4b82e286a659fae310012520f4f7
 56bb_normal.png",
                "created_at": "Thu Jan 18 00:10:45 +0000 2007", …
```

PARSING JSON IN PYTHON

Repeat: don't write your own CSV or JSON parser

- https://news.ycombinator.com/item?id=7796268
- rsdy.github.io/posts/dont_write_your_json_parser_plz.html

Python comes with a fine JSON parser

```
import json

r = requests.get(
"https://api.twitter.com/1.1/statuses/user_timeline.jso
n?screen_name=JohnPDickerson&count=100", auth=auth )

data = json.loads(r.content)
```

```
json.load(some_file)  # loads JSON from a file
json.dump(json_obj, some_file)  # writes JSON to file
json.dumps(json_obj)  # returns JSON string
```

XML, XHTML, HTML FILES AND STRINGS

Still hugely popular online, but JSON has essentially replaced XML for:

- Asynchronous browser ←→ server calls
- Many (most?) newer web APIs

XML is a hierarchical markup language:

```
<tag attribute="value1">
        <subtag>
                Some content goes here
        </subtag>
        <openclosetag attribute="value2" />
</tag>
```

You probably won't see much XML, but you will see plenty of HTML, is substantially less well-behaved cousin …

Example XML from: Zico Kolter

SCRAPING HTML IN PYTHON

HTML – the specification – is fairly pure

HTML – what you find on the web – is horrifying

We'll use BeautifulSoup:

- `conda install -c asmeurer beautiful-soup=4.3.2`

```
import requests
from bs4 import BeautifulSoup

r = requests.get(
"https://cs.umd.edu/class/spring2017/cmsc320/" )

root = BeautifulSoup( r.content )
root.find("div", id="schedule")\
    .find("table")\                  # find all schedule
    .find("tbody").findAll("a")  # links for CMSC320
```

BUILDING A WEB SCRAPER IN PYTHON

Totally not hypothetical situation:

- On May 20th, one day after turning in a Pulitzer-Prize-worthy mini-tutorial and receiving an A++ in CMSC320, you want to download all the lecture slides to wallpaper your room …
- … but you now have carpal tunnel syndrome from all that hard work, and can no longer click on the PDF and PPTX links.

Hopeless? No! Earlier, you built a scraper to do this!

```
lnks = root.find("div", id="schedule")\
    .find("table")\                  # find all schedule
    .find("tbody").findAll("a")  # links for CMSC320
```

Sort of. You only want PDF and PPTX files, not links to other websites or files.

REGULAR EXPRESSIONS

Given a list of URLs (strings), how do I find only those strings that end in *.pdf or *.pptx?

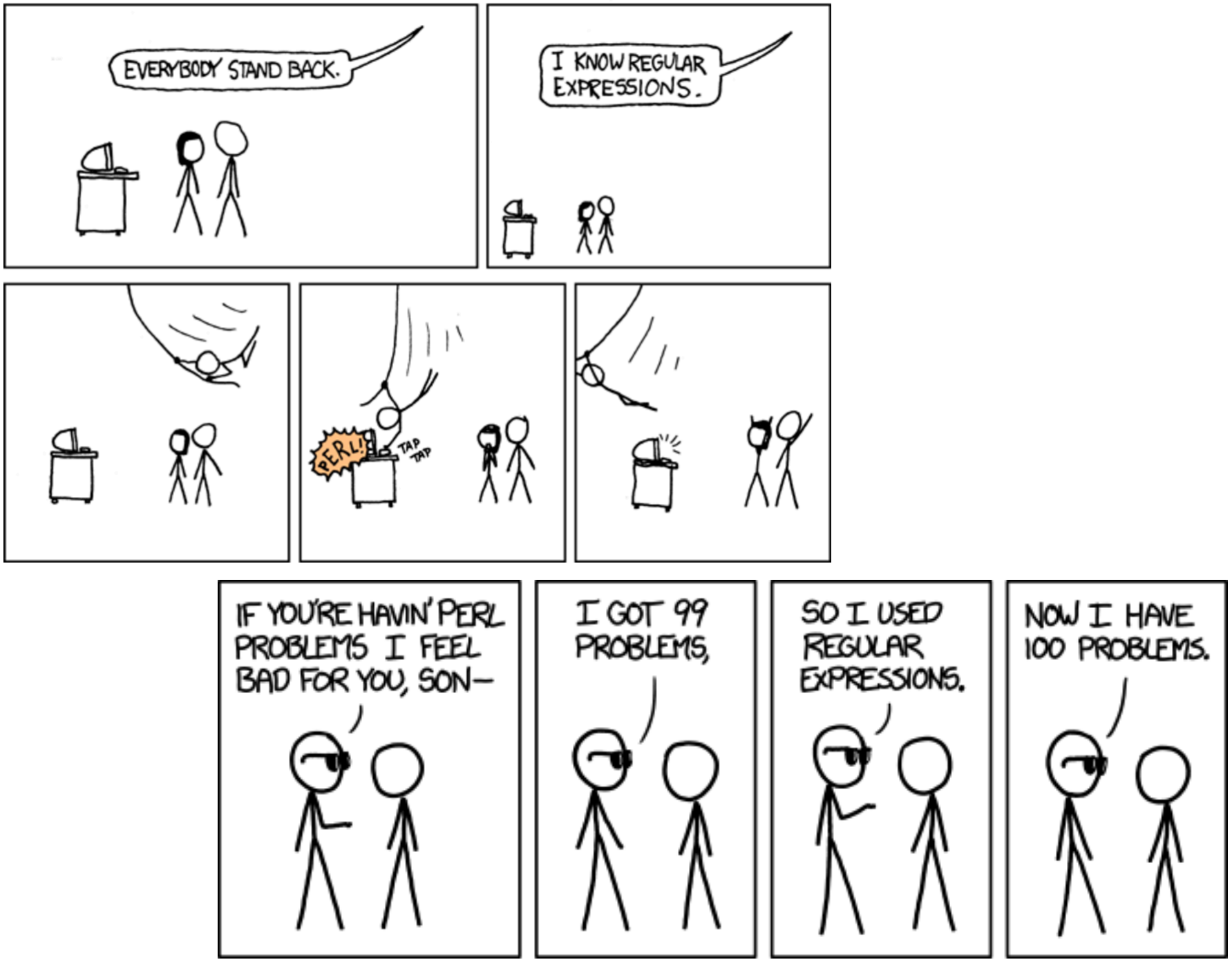
- Regular expressions!
- (Actually Python strings come with a built-in `endswith` function.)

```
“this_is_a_filename.pdf”.endswith((“.pdf”, “.pptx”))
```

What about .pDf or .pPTx, still legal extensions for PDF/PPTX?

- Regular expressions!
- (Or cheat the system again: built-in string `lower` function.)

```
“tHiS_IS_a_FileNAme.pDF”.lower().endswith(
                                    (“.pdf”, “.pptx”))
```


EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP
IF YOU'RE HAVIN' PERL PROBLEMS I FEEL BAD FOR YOU, SON—
I GOT 99 PROBLEMS,
SO I USED REGULAR EXPRESSIONS.
NOW I HAVE 100 PROBLEMS.

REGULAR EXPRESSIONS

Used to search for specific elements, or groups of elements, that match a pattern

```
import re

# Find the index of the 1st occurrence of "cmsc320"
match = re.search(r"cmsc320", text)
print( match.start() )
```

```
# Does start of text match "cmsc320"?
match = re.match(r"cmsc320", text)
```

```
# Iterate over all matches for "cmsc320" in text
for match in re.finditer(r"cmsc320", text):
    print( match.start() )
```

```
# Return all matches of "cmsc320" in the text
match = re.findall(r"cmsc320", text)
```

MATCHING MULTIPLE CHARACTERS

Can match sets of characters, or multiple and more elaborate sets and sequences of characters:

- Match the character ‘a’: `a`
- Match the character ‘a’, ‘b’, or ‘c’: `[abc]`
- Match any character except ‘a’, ‘b’, or ‘c’: `[^abc]`
- Match any digit: `\d` (= `[0123456789]` or `[0-9]`)
- Match any alphanumeric: `\w` (= `[a-zA-Z0-9_]`)
- Match any whitespace: `\s` (= `[ \t\n\r\f\v]`)
- Match any character: `.`

Special characters must be escaped: `.^$*+?{}\[]|()`

Thanks to: Zico Kolter

MATCHING SEQUENCES AND REPEATED CHARACTERS

A few common modifiers (available in Python and most other high-level languages; +, {n}, {n,} *may* not):

- Match character ‘a’ exactly once: `a`
- Match character ‘a’ zero or once: `a?`
- Match character ‘a’ zero or more times: `a*`
- Match character ‘a’ one or more times: `a+`
- Match character ‘a’ exactly *n* times: `a{n}`
- Match character ‘a’ at least n times: `a{n,}`

Example: match all instances of “University of <somewhere>” where <somewhere> is an alphanumeric string with at least 3 characters:

- `\s*University\sof\s\w{3,}`

COMPILED REGEXES

If you're going to reuse the same regex many times, or if you aren't but things are going slowly for some reason, try compiling the regular expression.

- https://blog.codinghorror.com/to-compile-or-not-to-compile/

```
# Compile the regular expression "cmsc320"
regex = re.compile(r"cmsc320")

# Use it repeatedly to search for matches in text
regex.match( text )    # does start of text match?
regex.search( text )   # find the first match or None
regex.findall( text )  # find all matches
```

Interested? CMSC330, CMSC430, CMSC452, talk to me.

DOWNLOADING A BUNCH OF FILES

Import the modules

```
import re
import requests
from bs4 import BeautifulSoup
try:
    from urllib.parse import urlparse
except ImportError:
    from urlparse import urlparse
```

Get some HTML via HTTP

```
# HTTP GET request sent to the URL url
r = requests.get( url )

# Use BeautifulSoup to parse the GET response
root = BeautifulSoup( r.content )
lnks = root.find("div", id="schedule")\
           .find("table")\
           .find("tbody").findAll("a")
```

DOWNLOADING A BUNCH OF FILES

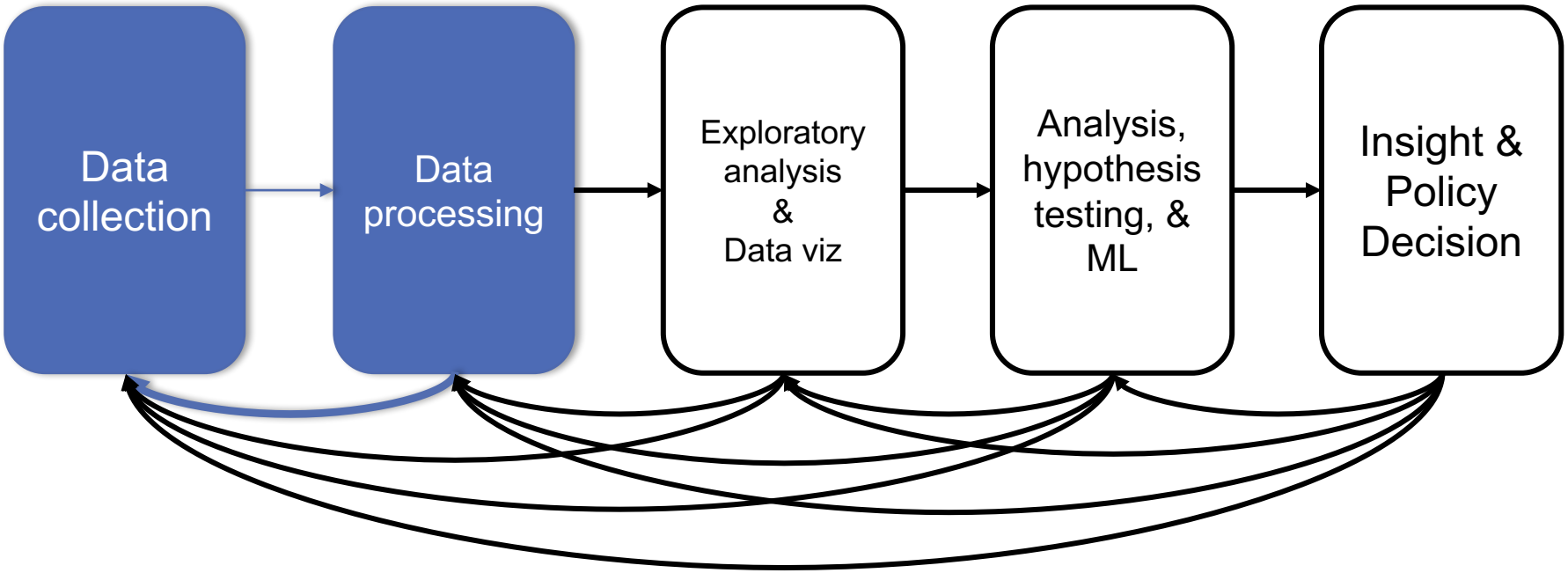
Parse exactly what you want

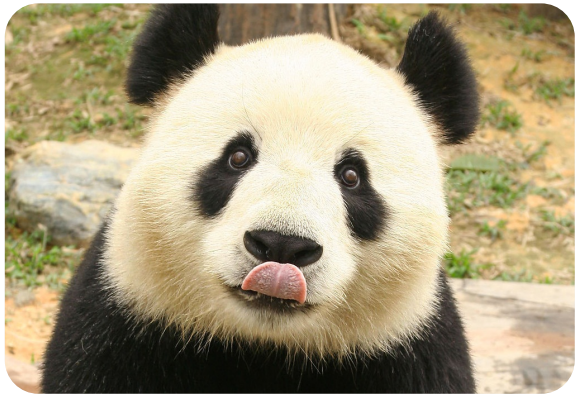
```
# Cycle through the href for each anchor, checking
# to see if it's a PDF/PPTX link or not
for lnk in lnks:
    href = lnk['href']

    # If it's a PDF/PPTX link, queue a download
    if href.lower().endswith(('.pdf', '.pptx')):
```

Get some more data?!

```
        urld = urlparse.urljoin(url, href)
        rd = requests.get(urld, stream=True)

        # Write the downloaded PDF to a file
        outfile = path.join(outbase, href)
        with open(outfile, 'wb') as f:
            f.write(rd.content)
```

NEXT LECTURE





NEXT CLASS:

VECTORS, MATRICES, AND DATAFRAMES

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$