

PRINCIPLES OF DATA SCIENCE

JOHN P DICKERSON

Lecture #2 – 9/5/2018

CMSC641
Wednesdays
7pm – 9:30pm



COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

ANNOUNCEMENTS

Register on Piazza: piazza.com/umd/fall2018/cmssc641

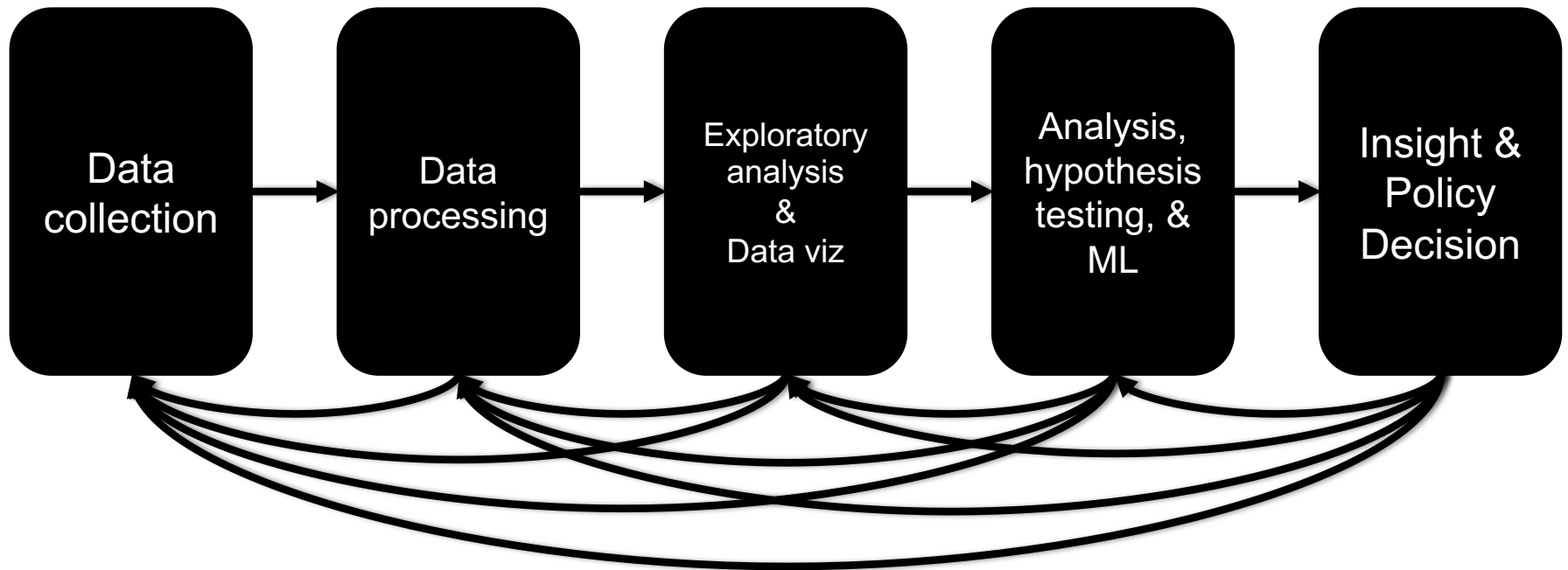
- I am a hypocrite! Somehow wasn't getting notifications – I will be much more responsive from here on out.

Office Hours: 6pm-7pm on Wednesdays, and also by appointment, or via email/text, or ...

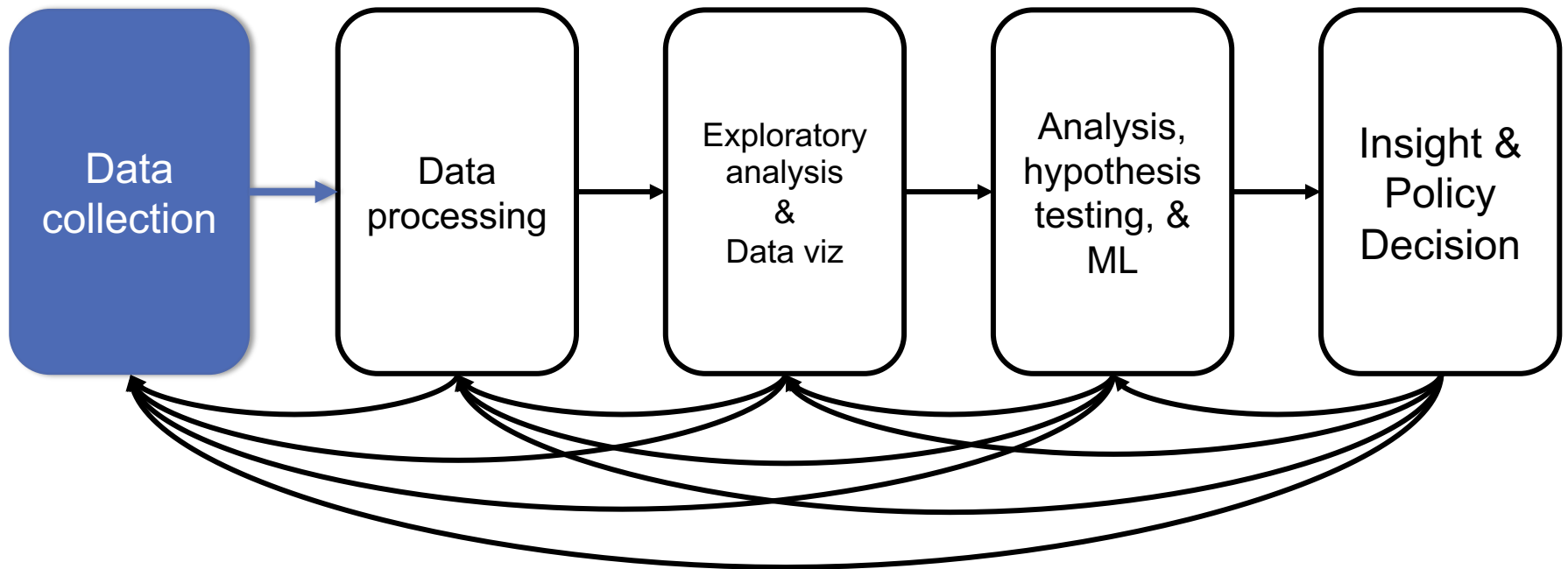
Reminder: Weekly quizzes, due on Mondays at noon (except first one that was due today)

Project 1 will be released soon (early next week)

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!**

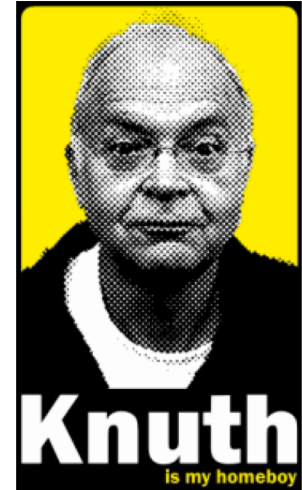
*you can compile Python source, but it's not required

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.



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.

Stuff you'll be using in Project 1 (and beyond)!

IP[y]: IPython
Interactive Computing



Jupyter

JUPYTER PROJECT

Started as iPython Notebooks, a web-based frontend to the iPython Shell

- Notebook functionality separated out a few years ago
- Now supports over 40 languages/kernels
- Notebooks can be shared easily
- Can leverage big data tools like Spark

Apache Zeppelin:

- <https://www.linkedin.com/pulse/comprehensive-comparison-jupyter-vs-zeppelin-hoc-q-phan-mba->

Several others including RStudio (specific to R)

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!

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

Construct lists like a mathematician **who codes!**

```
P = [ 2**x 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”)

*follow
your*



© Matplotlib

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
```

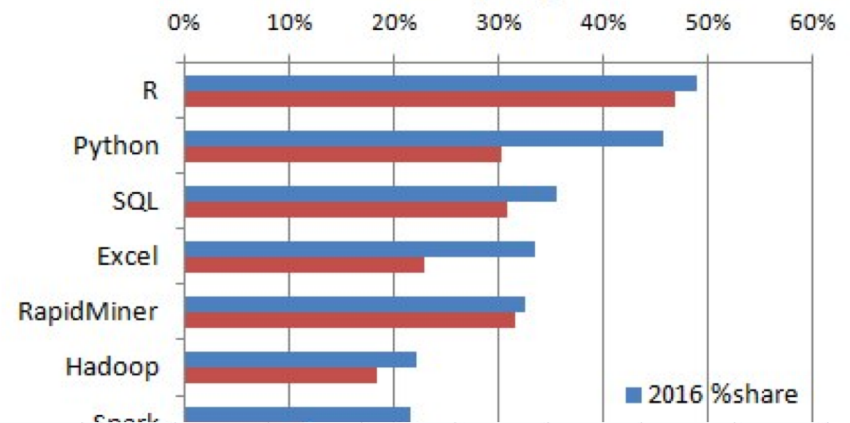
PYTHON VS R (FOR DATA SCIENTISTS)

There is no right answer here!

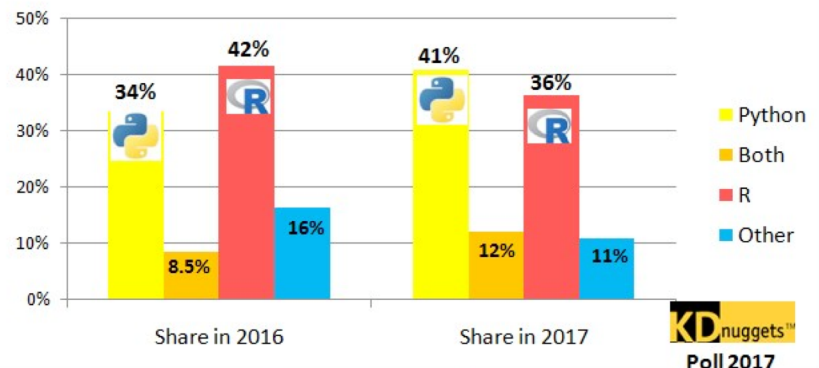
- Python is a “full” programming language – easier to integrate with systems in the field
- R has a more mature set of pure stats libraries ...
- ... but Python is catching up quickly ...
- ... and is already ahead **specifically for ML.**

You will see Python more in the tech industry.

KDnuggets Analytics/Data Science
2016 Software Poll, top 10 tools



Python, R, Both, or Other platforms for
Analytics, Data Science, Machine Learning



EXTRA RESOURCES

Plenty of tutorials on the web:

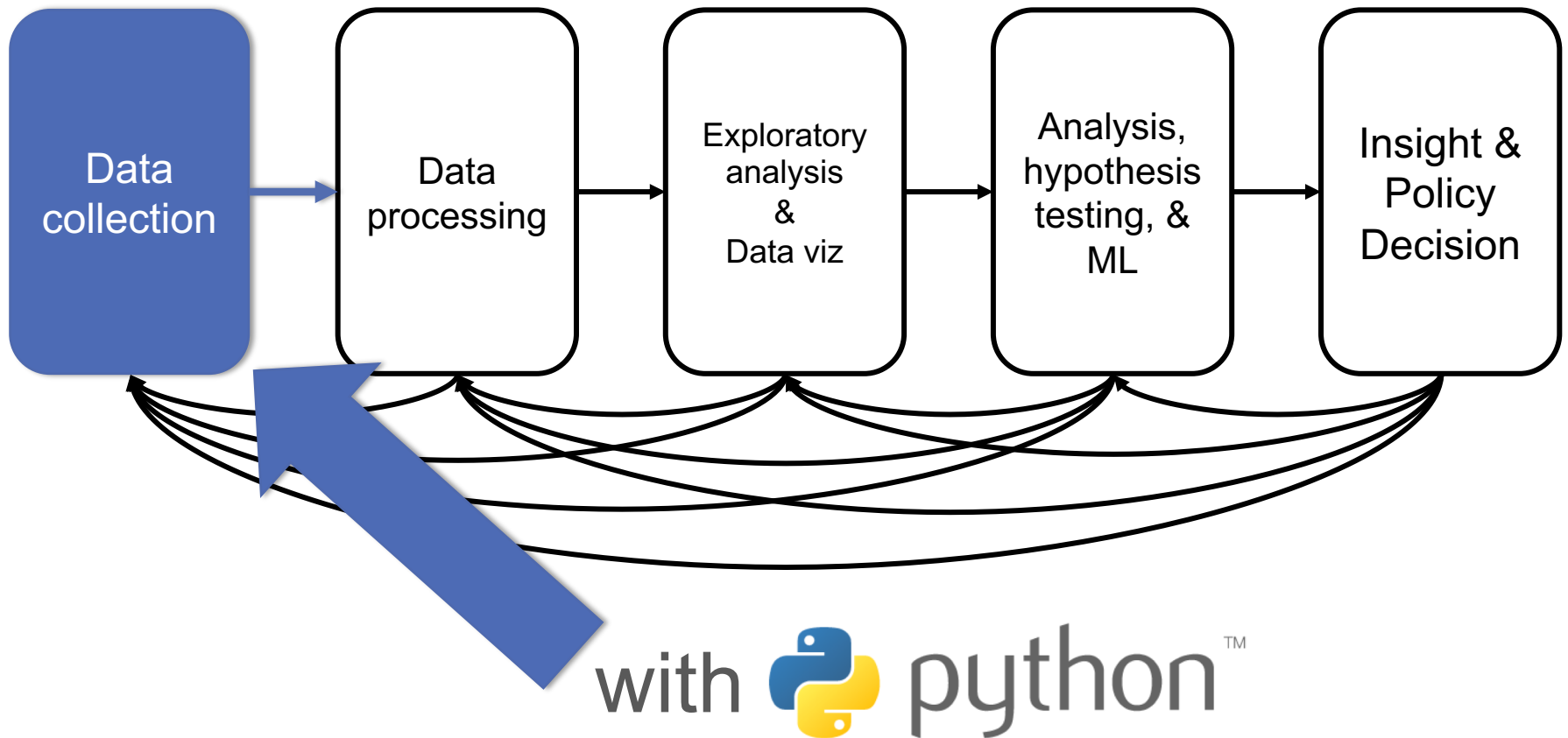
- <https://www.learnpython.org/>

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

- Office hours are on the website/Piazza very soon.
- Also, email me – I realize your schedules are not like undergrads' schedules 😊.



TODAY'S LECTURE



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 in a few lectures)
- Query an API from the intra/internet
- Scrape data from a webpage



Covered today.

WHEREFORE ART THOU, API?

A web-based **A**pplication **P**rogramming **I**nterface (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 REQUESTS

`https://www.google.com/?q=cmssc641&tbs=qdr:m`



??????????

HTTP GET Request:

GET `/?q=cmssc641&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": "cmssc641", "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.*

GRAPHQL?

An alternative to REST and ad-hoc webservice architectures


- Developed internally by Facebook and released publicly

Unlike REST, the requester specifies the format of the response



```
GET /books/1
```

```
{
  "title": "Black Hole Blues",
  "author": {
    "firstName": "Janna",
    "lastName": "Levin"
  }
  // ... more fields here
}
```



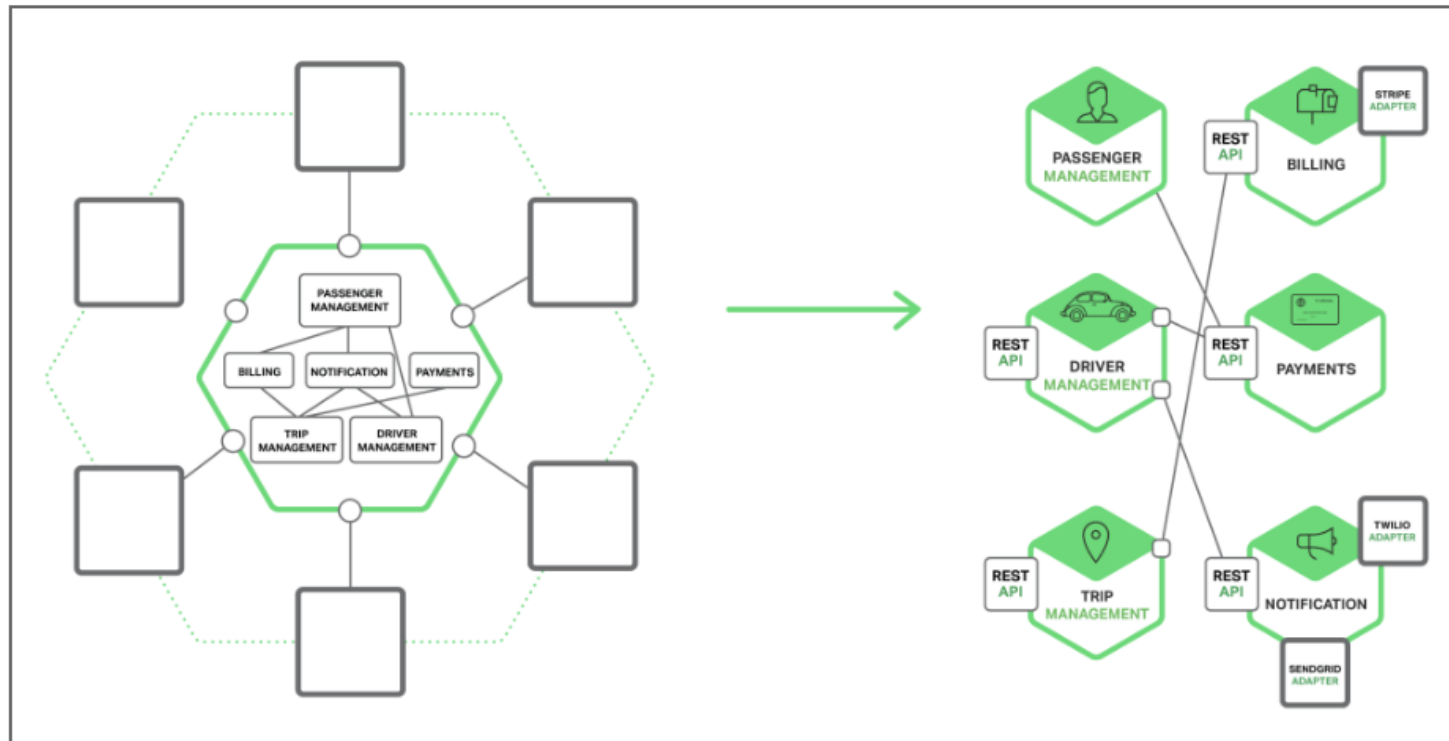
```
GET /graphql?query={ book(id: "1") { title, author { firstName } } }
```

```
{
  "title": "Black Hole Blues",
  "author": {
    "firstName": "Janna",
  }
}
```

APIS AND MICROSERVICES

Use of APIs growing rapidly, especially because of increasing trend towards “microservices”

OpenAPI (Swagger) Specification emerging as a standard for



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)

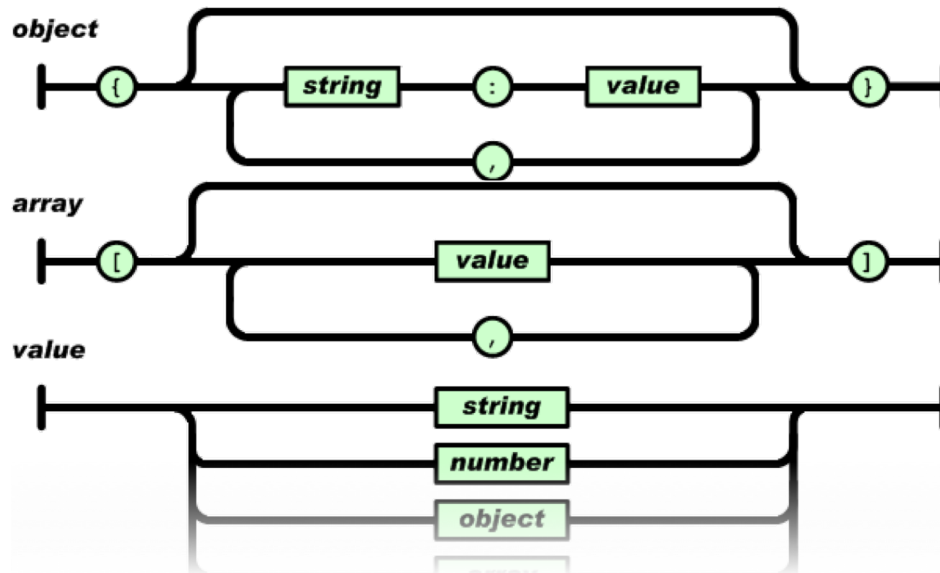
JSON FILES & STRINGS

JSON is a method for **serializing** objects:

- Convert object into a string (Java: “implements Serializable”)
- **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"},
    "CMSC641"
}]}
```



JSON FROM TWITTER

```
GET https://api.twitter.com/1.1/friends/list.json?cursor=-1&screen_name=twitterapi&skip_status=true&include_user_entities=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/2838630046/4b82e286a659fae310012520f4f756bb_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.json?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:

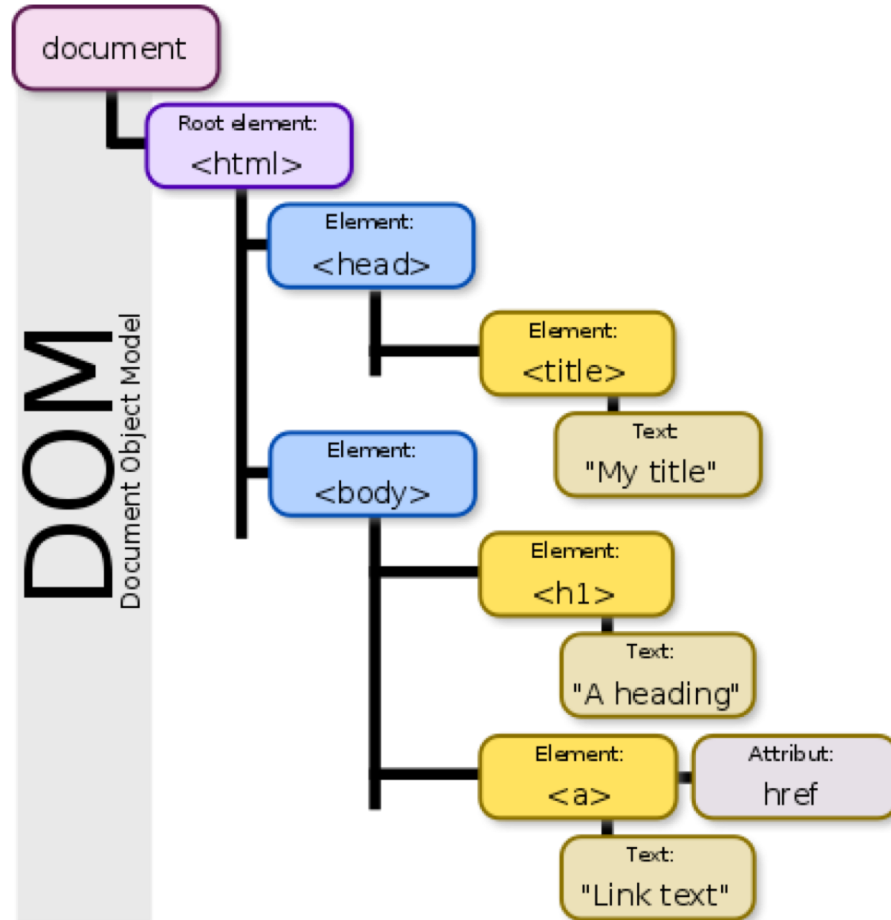
- Asynchronous browser \leftrightarrow server calls
- Many (most?) newer web APIs

XML is a hierarchical markup language:

```
<tag attribute="value1">
    <subtag>
        Some cool words or values go here!
    </subtag>
    <openclosetag attribute="value2" />
</tag>
```

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

DOCUMENT OBJECT MODEL (DOM)



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/fall2018/cmsc641/" )

root = BeautifulSoup( r.content )
root.find("div", id="schedule")\
    .find("table")\                                # find all schedule
    .find("tbody").findAll("a")                    # links for CMSC641
```

BUILDING A WEB SCRAPER IN PYTHON

Totally not hypothetical situation:

- You really want to learn about data science, so you choose to download all of this year's CMSC641 lecture slides to wallpaper your room ...
- ... but you now have carpal tunnel syndrome from clicking refresh on Piazza last night, 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 CMSC641
```

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?

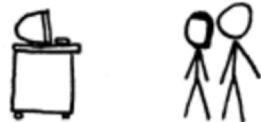
- Regular expressions!
- (Or cheat the system again: built-in string `lower` function.)

```
"tHiS_IS_a_FiLeNaMe.pDF".lower().endswith(
    (".pdf", ".pptx"))
```

EVERYBODY STAND BACK.



I KNOW REGULAR EXPRESSIONS.



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 "cmssc641"  
match = re.search(r"cmssc641", text)  
print( match.start() )
```

```
# Does start of text match "cmssc641"?  
match = re.match(r"cmssc641", text)
```

```
# Iterate over all matches for "cmssc641" in text  
for match in re.finditer(r"cmssc641", text):  
    print( match.start() )
```

```
# Return all matches of "cmssc641" in the text  
match = re.findall(r"cmssc641", 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: `.^$*+?{\} \[] | ()`

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\s of\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 "cmssc320"
regex = re.compile(r"cmssc320")

# 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? CMSC6*, CMSC7*, CMSC8*, talk to me.

GROUPS

What if we want to know more than just “did we find a match” or “where is the first match” ...?

Grouping asks the regex matcher to keep track of certain portions – surrounded by (parentheses) – of the match

```
\s*([Uu]niversity)\s([Oo]f)\s(\w{3,})
```

```
regex = r"\s*([Uu]niversity)\s([Oo]f)\s(\w{3,})"  
m = re.search( regex, "university Of Maryland" )  
print( m.groups() )
```

```
('university', 'Of', 'Maryland')
```

47

Mail::RFC822::Address Perl module for RFC 822

NAMED GROUPS

Raw grouping is useful for one-off exploratory analysis, but may get confusing with longer regexes

- Much scarier regexes than that email one exist in the wild ...

Named groups let you attach position-independent identifiers to groups in a regex

```
(?P<some_name> ...)
```

```
regex = "\s*[Uu]niversity\s[Oo]f\s(?P<school>(\w{3,}))"  
m = re.search( regex, "University of Maryland" )  
print( m.group('school') )
```

```
'Maryland'
```

SUBSTITUTIONS

The Python `string` module contains basic functionality for find-and-replace within strings:

```
"abcabcabc".replace("a", "X")
```

```
`XbcXbcXbc`
```

For more complicated stuff, use regexes:

```
text = "I love Principles of Data Science"  
re.sub(r"Data Science", r"Schmada Schmience", text)
```

```
`I love Principles of Schmada Schmience`
```

Can incorporate groups into the matching

```
re.sub(r"(\w+)\s([Ss])cience", r"\1 \2hmience", text)
```

```
re.sub(r"([a-zA-Z0123456789_]+)\s([Ss])cience", r"\1  
\2hmience", text)
```

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

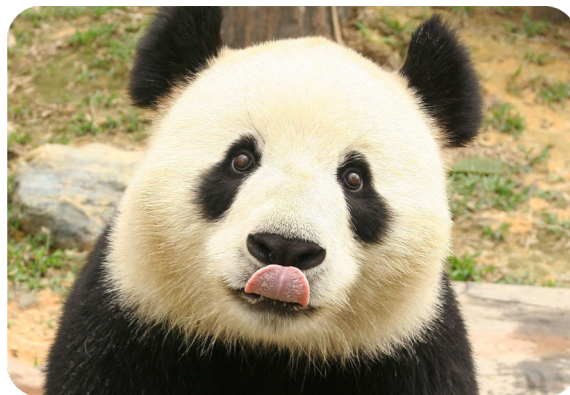
```
# Cycle through the href for each anchor, checking
# to see if it's a PDF/PPTX link or not
for lnk in lnks:
    href = lnk['href']

    # 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)

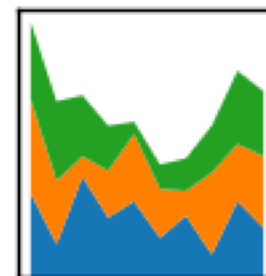
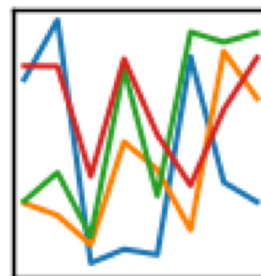
# Write the downloaded PDF to a file
outfile = path.join(outbase, href)
with open(outfile, 'wb') as f:
    f.write(rd.content)
```



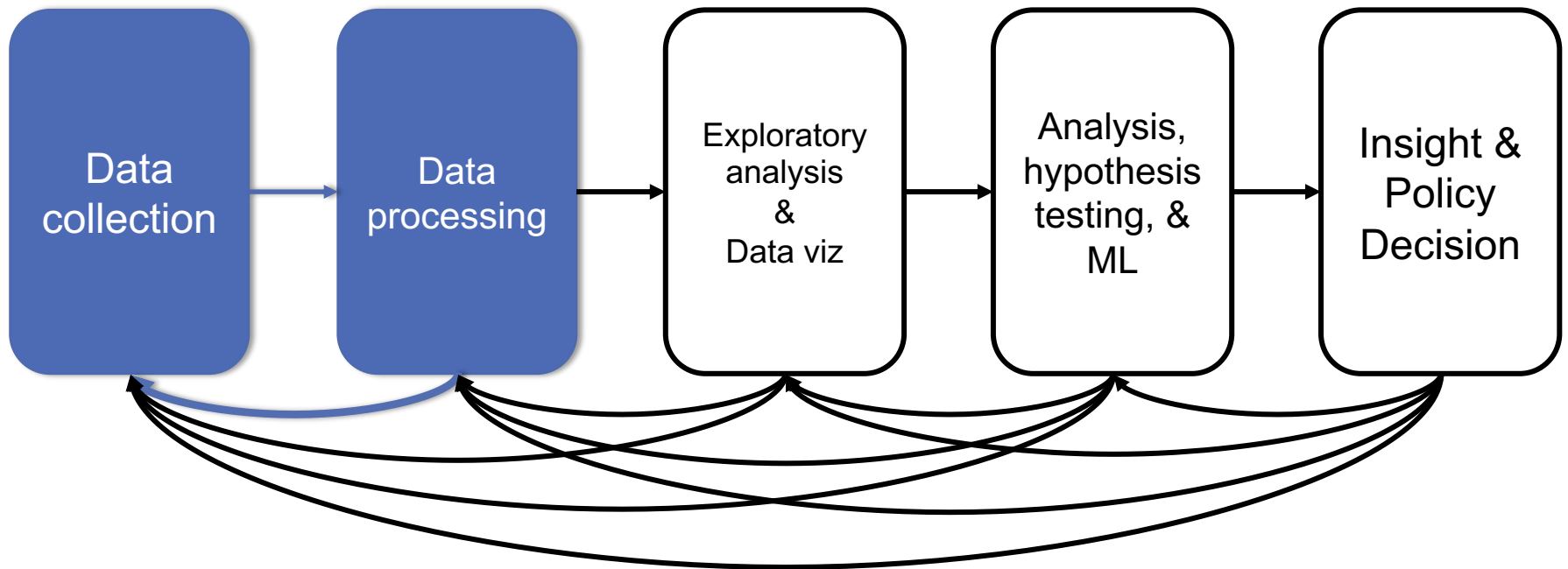
AFTER THE BREAK, AND SOME OF NEXT CLASS:
NUMPY, SCIPY, AND DATAFRAMES

pandas

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



THE DATA LIFECYCLE



DATA MANIPULATION AND COMPUTATION

Data Science == manipulating and computing on data

Large to very large, but somewhat “structured” data

We will see several tools for doing that this semester

Thousands more out there that we won't cover

Need to learn to shift thinking from:

Imperative code to manipulate data structures

to:

Sequences/pipelines of operations on data

Should still know how to implement the operations themselves, especially for debugging performance (covered in classes like 642?, 643?), but we won't cover that much

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

One-dimensional Arrays, Vectors

0.1	2	3.2	6.5	3.4	4.1
-----	---	-----	-----	-----	-----

"data"	"representation"	"i.e."
--------	------------------	--------

Indexing

Slicing/subsetting

Filter

'map' → apply a function to every element

'reduce/aggregate' → combine values to get a single scalar (e.g., sum, median)

Given two vectors: **Dot and cross products**

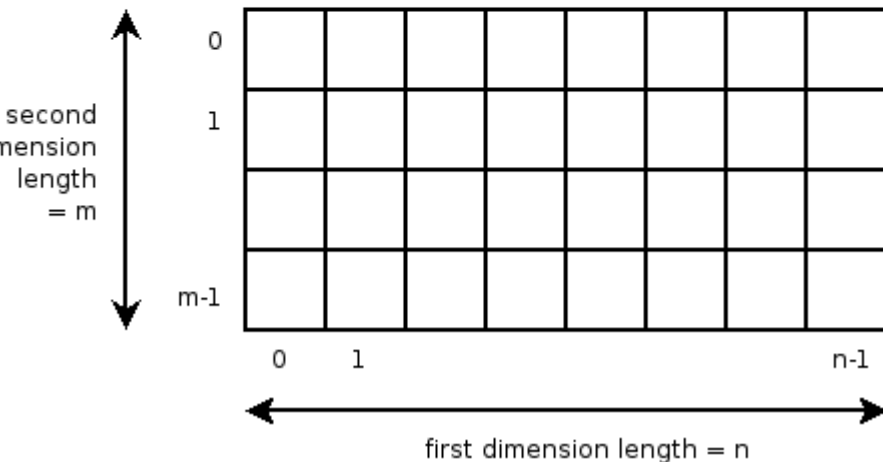
2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

n-dimensional arrays

Two-dimensional array



Indexing

Slicing/subsetting

Filter

'map' → apply a function to every element

'reduce/aggregate' → combine values across a row or a column (e.g., sum, average, median etc..)

2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

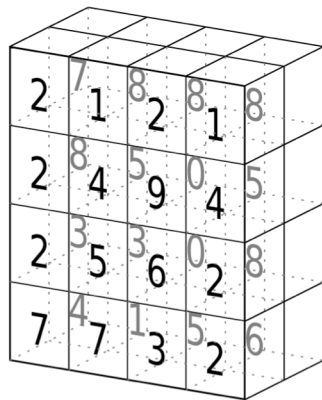
DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

Matrices, Tensors

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3
2	3	8	4
6	2	6	4

tensor of dimensions [6,4]
(matrix 6 by 4)



tensor of dimensions [4,4,2]

n-dimensional array operations
+

Linear Algebra

Matrix/tensor multiplication

Transpose

Matrix-vector multiplication

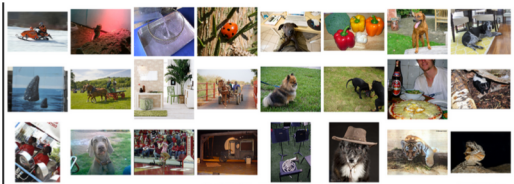
Matrix factorization

2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

Sets: of Objects



Filter
Map
Union

Reduce/Aggregate

Sets: of (Key, Value Pairs)

(amol@cs.umd.edu, (email1, email2, ...))

(john@cs.umd.edu, (email3, email4, ...))

Given two sets, **Combine/Join** using “keys”

Group and then aggregate

2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

Tables/Relations == Sets of Tuples

company	division	sector	tryint
00nil_Combined_Company	00nil_Combined_Division	00nil_Combined_Sector	14625
apple	00nil_Combined_Division	00nil_Combined_Sector	10125
apple	hardware	00nil_Combined_Sector	4500
apple	hardware	business	1350
apple	hardware	consumer	3150
apple	software	00nil_Combined_Sector	5625
apple	software	business	4950
apple	software	consumer	675
microsoft	00nil_Combined_Division	00nil_Combined_Sector	4500
microsoft	hardware	00nil_Combined_Sector	1890
microsoft	hardware	business	855
microsoft	hardware	consumer	1035
microsoft	software	00nil_Combined_Sector	2610
microsoft	software	business	1215
microsoft	software	consumer	1395

Filter rows or columns

”Join” two or more relations

”Group” and “aggregate” them

Relational Algebra formalizes some of them

Structured Query Language (SQL)

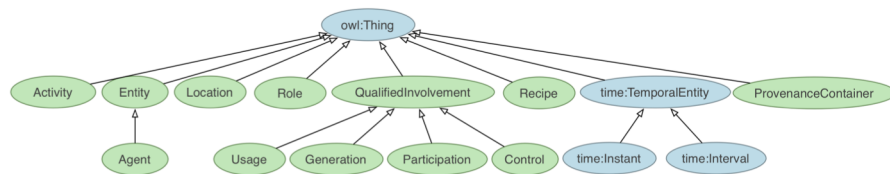
Many other languages and constructs, that look very similar

2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

Hierarchies/Trees/Graphs



"Path" queries

Graph Algorithms and Transformations

Network Science

Somewhat more ad hoc and special-purpose

Changing in recent years

2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data
2. **Data Processing Operations**, which take one or more datasets as input and produce
 - **Why?**
 - Allows one to think at a higher level of abstraction, leading to simpler and easier-to-understand scripts
 - Provides "independence" between the abstract operations and concrete implementation
 - Can switch from one implementation to another easily
 - **For performance debugging, useful to know how they are implemented and rough characteristics**

NEXT COUPLE OF CLASSES

1. **NumPy: Python Library for Manipulating nD Arrays**

Multidimensional Arrays, and a variety of operations including Linear Algebra

2. **Pandas: Python Library for Manipulating Tabular Data**

Series, Tables (also called **DataFrames**)

Many operations to manipulate and combine tables/series

3. **Relational Databases**

Tables/Relations, and SQL (similar to Pandas operations)

4. **Apache Spark**

Sets of objects or key-value pairs

MapReduce and SQL-like operations

NEXT COUPLE OF CLASSES

1. NumPy: Python Library for Manipulating nD Arrays

Multidimensional Arrays, and a variety of operations including Linear Algebra

2. Pandas: Python Library for Manipulating Tabular Data

Series, Tables (also called **DataFrames**)

Many operations to manipulate and combine tables/series

3. Relational Databases

Tables/Relations, and SQL (similar to Pandas operations)

4. Apache Spark

Sets of objects or key-value pairs

MapReduce and SQL-like operations

NUMERIC & SCIENTIFIC APPLICATIONS

Number of third-party packages available for numerical and scientific computing

These include:

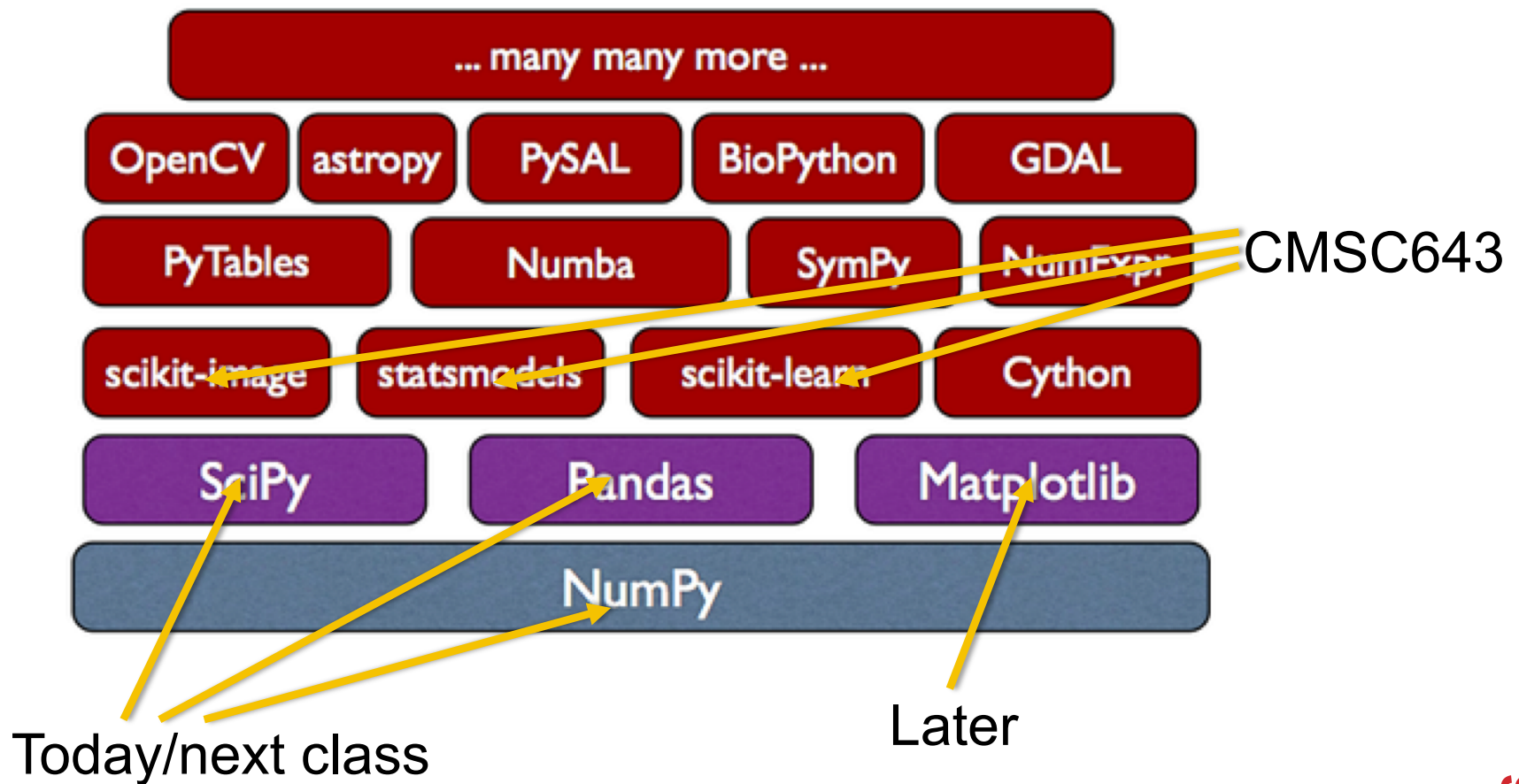
- NumPy/SciPy – numerical and scientific function libraries.
- numba – Python compiler that support JIT compilation.
- ALGLIB – numerical analysis library.
- pandas – high-performance data structures and data analysis tools.
- pyGSL – Python interface for GNU Scientific Library.
- ScientificPython – collection of scientific computing modules.

NUMPY AND FRIENDS

By far, the most commonly used packages are those in the NumPy stack. These packages include:

- NumPy: similar functionality as Matlab
- SciPy: integrates many other packages like NumPy
- Matplotlib & Seaborn – plotting libraries
- iPython via Jupyter – interactive computing
- Pandas – data analysis library
- SymPy – symbolic computation library

THE NUMPY STACK



NUMPY

Among other things, NumPy contains:

- A powerful n -dimensional array object.
- Sophisticated (broadcasting/universal) functions.
- Tools for integrating C/C++ and Fortran code.
- Useful linear algebra, Fourier transform, and random number capabilities, etc.

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.



NUMPY

ndarray object: an n -dimensional array of homogeneous data types, with many operations being performed in compiled code for performance

Several important differences between NumPy arrays and the standard Python sequences:

- NumPy arrays have a fixed size. Modifying the size means creating a new array.
- NumPy arrays must be of the same data type, but this can include Python objects – may not get performance benefits
- More efficient mathematical operations than built-in sequence types.

NUMPY DATATYPES

Wider variety of data types than are built-in to the Python language by default.

Defined by the `numpy.dtype` class and include:

- `intc` (same as a C integer) and `intp` (used for indexing)
- `int8`, `int16`, `int32`, `int64`
- `uint8`, `uint16`, `uint32`, `uint64`
- `float16`, `float32`, `float64`
- `complex64`, `complex128`
- `bool_`, `int_`, `float_`, `complex_` are shorthand for defaults.

These can be used as functions to cast literals or sequence types, as well as arguments to NumPy functions that accept the `dtype` keyword argument.

NUMPY DATATYPES

```
>>> import numpy as np
>>> x = np.float32(1.0)
>>> x
1.0
>>> y = np.int_([1,2,4])
>>> y
array([1, 2, 4])
>>> z = np.arange(3, dtype=np.uint8)
>>> z
array([0, 1, 2], dtype=uint8)
>>> z.dtype
dtype('uint8')
```

NUMPY ARRAYS

There are a couple of mechanisms for creating arrays in NumPy:

- Conversion from other Python structures (e.g., lists, tuples)
 - Any sequence-like data can be mapped to a ndarray
- Built-in NumPy array creation (e.g., `arange`, `ones`, `zeros`, etc.)
 - Create arrays with all zeros, all ones, increasing numbers from 0 to 1 etc.
- Reading arrays from disk, either from standard or custom formats (e.g., reading in from a CSV file)

NUMPY ARRAYS

In general, any numerical data that is stored in an array-like container can be converted to an `ndarray` through use of the `array()` function. The most obvious examples are sequence types like lists and tuples.

```
>>> x = np.array([2,3,1,0])
```

```
>>> x = np.array([2, 3, 1, 0])
```

```
>>> x = np.array([[1,2.0],[0,0]],[1+1j,3.])])
```

```
>>> x = np.array([[ 1.+0.j, 2.+0.j], [ 0.+0.j, 0.+0.j],  
[ 1.+1.j, 3.+0.j]])
```

NUMPY ARRAYS

Creating arrays from scratch in NumPy:

- `zeros(shape)` – creates an array filled with 0 values with the specified shape. The default `dtype` is `float64`.

```
>>> np.zeros((2, 3))  
array([[ 0.,  0.,  0.], [ 0.,  0.,  0.]])
```

- `ones(shape)` – creates an array filled with 1 values.
- `arange()` – like Python's built-in `range`

```
>>> np.arange(10)  
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])  
>>> np.arange(2, 10, dtype=np.float)  
array([ 2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])  
>>> np.arange(2, 3, 0.2)  
array([ 2. ,  2.2,  2.4,  2.6,  2.8])
```

NUMPY ARRAYS

linspace() – creates arrays with a specified number of elements, and spaced equally between the specified beginning and end values.

```
>>> np.linspace(1., 4., 6)
array([ 1. , 1.6, 2.2, 2.8, 3.4, 4. ])
```

random.random(shape) – creates arrays with random floats over the interval [0,1).

```
>>> np.random.random((2,3))
array([[ 0.75688597,  0.41759916,  0.35007419],
       [ 0.77164187,  0.05869089,  0.98792864]])
```

NUMPY ARRAYS

Printing an array can
be done with the print

- statement (Python 2)
- function (Python 3)

```
>>> import numpy as np
>>> a = np.arange(3)
>>> print(a)
[0 1 2]
>>> a
array([0, 1, 2])
>>> b = np.arange(9).reshape(3,3)
>>> print(b)
[[0 1 2]
 [3 4 5]
 [6 7 8]]
>>> c =
np.arange(8).reshape(2,2,2)
>>> print(c)
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
```

INDEXING

Single-dimension indexing is accomplished as usual.

```
>>> x = np.arange(10)
>>> x[2]
2
>>> x[-2]
8
```

Multi-dimensional arrays support multi-dimensional indexing.

```
>>> x.shape = (2,5) # now x is 2-dimensional
>>> x[1,3]
8
>>> x[1,-1]
9
```


INDEXING

Using fewer dimensions to index will result in a subarray:

```
>>> x = np.arange(10)
>>> x.shape = (2,5)
>>> x[0]
array([0, 1, 2, 3, 4])
```

This means that $x[i, j] == x[i][j]$ but the second method is less efficient.

INDEXING

Slicing is possible just as it is for typical Python sequences:

```
>>> x = np.arange(10)
>>> x[2:5]
array([2, 3, 4])
>>> x[: -7]
array([0, 1, 2])
>>> x[1:7:2]
array([1, 3, 5])
>>> y = np.arange(35).reshape(5,7)
>>> y[1:5:2, ::3]
array([[ 7, 10, 13], [21, 24, 27]])
```

ARRAY OPERATIONS

Basic operations apply element-wise. The result is a new array with the resultant elements.

```
>>> a = np.arange(5)
>>> b = np.arange(5)
>>> a+b
array([0, 2, 4, 6, 8])
>>> a-b
array([0, 0, 0, 0, 0])
>>> a**2
array([ 0,  1,  4,  9, 16])
>>> a>3
array([False, False, False, False,  True], dtype=bool)
>>> 10*np.sin(a)
array([ 0.,  8.41470985,  9.09297427,  1.41120008, -
 7.56802495])
>>> a*b
array([ 0,  1,  4,  9, 16])
```

ARRAY OPERATIONS

Since multiplication is done element-wise, you need to specifically perform a dot product to perform matrix multiplication.

```
>>> a = np.zeros(4).reshape(2,2)
>>> a
array([[ 0.,  0.],
       [ 0.,  0.]])
>>> a[0,0] = 1
>>> a[1,1] = 1
>>> b = np.arange(4).reshape(2,2)
>>> b
array([[0, 1],
       [2, 3]])
>>> a*b
array([[ 0.,  0.],
       [ 0.,  3.]])
>>> np.dot(a,b)
array([[ 0.,  1.],
       [ 2.,  3.]])
```

ARRAY OPERATIONS

There are also some built-in methods of ndarray objects.

Universal functions which may also be applied include `exp`, `sqrt`, `add`, `sin`, `cos`, etc.

```
>>> a = np.random.random((2,3))
>>> a
array([[ 0.68166391,  0.98943098,
         0.69361582],
       [ 0.78888081,  0.62197125,
         0.40517936]])
>>> a.sum()
4.1807421388722164
>>> a.min()
0.4051793610379143
>>> a.max(axis=0)
array([ 0.78888081,  0.98943098,
        0.69361582])
>>> a.min(axis=1)
array([ 0.68166391,  0.40517936])
```

ARRAY OPERATIONS

An array shape
can be
manipulated by a
number of
methods.

`resize(size)`
will modify an
array in place.

`reshape(size)`
will return a copy
of the array with a
new shape.

```
>>> a =  
np.floor(10*np.random.random((3,4)))  
>>> print(a)  
[[ 9.  8.  7.  9.]  
 [ 7.  5.  9.  7.]  
 [ 8.  2.  7.  5.]]  
>>> a.shape  
(3, 4)  
>>> a.ravel()  
array([ 9.,  8.,  7.,  9.,  7.,  5.,  9.,  
        7.,  8.,  2.,  7.,  5.])  
>>> a.shape = (6,2)  
>>> print(a)  
[[ 9.  8.]  
 [ 7.  9.]  
 [ 7.  5.]  
 [ 9.  7.]  
 [ 8.  2.]  
 [ 7.  5.]]  
>>> a.transpose()  
array([[ 9.,  7.,  7.,  9.,  8.,  7.],  
       [ 8.,  9.,  5.,  7.,  2.,  5.]])
```

LINEAR ALGEBRA

One of the most common reasons for using the NumPy package is its linear algebra module.

It's like Matlab, but free!

```
>>> from numpy import *
>>> from numpy.linalg import *
>>> a = array([[1.0, 2.0],
               [3.0, 4.0]])

>>> print(a)
[[ 1.  2.]
 [ 3.  4.]]
>>> a.transpose()
array([[ 1.,  3.],
       [ 2.,  4.]])
>>> inv(a) # inverse
array([[ -2. ,  1. ],
       [ 1.5, -0.5]])
```

LINEAR ALGEBRA

```
>>> u = eye(2) # unit 2x2 matrix; "eye" represents "I"
>>> u
array([[ 1.,  0.],
       [ 0.,  1.]])
>>> j = array([[0.0, -1.0], [1.0, 0.0]])
>>> dot(j, j) # matrix product
array([[ -1.,  0.],
       [ 0., -1.]])
>>> trace(u) # trace (sum of elements on diagonal)
2.0
>>> y = array([[5.], [7.]])
>>> solve(a, y) # solve linear matrix equation
array([[ -3.],
       [ 4.]])
>>> eig(j) # get eigenvalues/eigenvectors of matrix
(array([ 0.+1.j, 0.-1.j]),
 array([[ 0.70710678+0.j, 0.70710678+0.j],
        [ 0.00000000-0.70710678j,
         0.00000000+0.70710678j]]))
```


SCIPY?



In its own words:

SciPy is a collection of mathematical algorithms and convenience functions **built on the NumPy extension** of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data.

Basically, SciPy contains various tools and functions for solving common problems in **scientific computing.**

SCIPY

SciPy gives you access to a ton of specialized mathematical functionality.

- **Just know it exists.** We won't use it much in this class.

Some functionality:

- Special mathematical functions (`scipy.special`) -- elliptic, *bessel*, etc.
- Integration (`scipy.integrate`)
- Optimization (`scipy.optimize`)
- Interpolation (`scipy.interpolate`)
- Fourier Transforms (`scipy.fftpack`)
- Signal Processing (`scipy.signal`)
- Linear Algebra (`scipy.linalg`)
- Compressed Sparse Graph Routines (`scipy.sparse.csgraph`)
- Spatial data structures and algorithms (`scipy.spatial`)
- Statistics (`scipy.stats`)
- Multidimensional image processing (`scipy.ndimage`)
- Data IO (`scipy.io`) – overlaps with *pandas*, covers some other formats

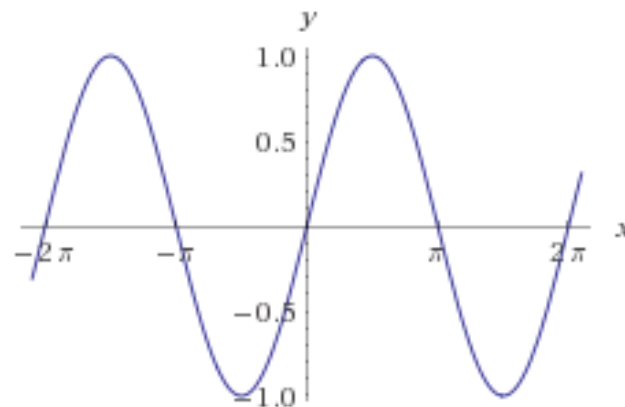
ONE SCIPY EXAMPLE

We can't possibly tour all of the SciPy library and, even if we did, it might be a little boring.

- Often, you'll be able to find higher-level modules that will work around your need to directly call low-level SciPy functions

Say you want to compute an integral:

$$\int_a^b \sin x \, dx$$



SCIPY.INTEGRATE

We have a function object – `np.sin` defines the sin function for us.

We can compute the definite integral from $x = 0$ to $x = \pi$ using the quad function.

```
>>> res = scipy.integrate.quad(np.sin, 0, np.pi)
>>> print(res)
(2.0, 2.220446049250313e-14) # 2 with a very small error
margin!
>>> res = scipy.integrate.quad(np.sin, -np.inf, +np.inf)
>>> print(res)
(0.0, 0.0) # Integral does not converge
```

SCIPY.INTEGRATE

Let's say that we don't have a function object, we only have some (x,y) samples that “define” our function.

We can estimate the integral using the trapezoidal rule.

```
>>> sample_x = np.linspace(0, np.pi, 1000)
>>> sample_y = np.sin(sample_x) # Creating 1,000 samples
>>> result = scipy.integrate.trapz(sample_y, sample_x)
>>> print(result)
1.99999835177
```

```
>>> sample_x = np.linspace(0, np.pi, 1000000)
>>> sample_y = np.sin(sample_x) # Creating 1,000,000
samples
>>> result = scipy.integrate.trapz(sample_y, sample_x)
>>> print(result)
2.0
```

WRAP UP

Shift thinking from imperative coding to operations on datasets

Numpy: A low-level abstraction that gives us really fast multi-dimensional arrays

Next class:

Pandas: Higher-level tabular abstraction and operations to manipulate and combine tables

Reading Homework focuses on Pandas and SQL: Aim to release by tonight, probably by tomorrow

REST OF TODAY'S LECTURE

By popular request ...

- **Version control** primer!
- Specifically, git via GitHub and GitLab
- Thanks: Mark Groves (Microsoft), Ilan Biala & Aaron Perley (CMU), Sharif U., & the HJCB Senior Design Team!

And then a bit on keeping your data ... **tidy data**.



WHAT IS VERSION CONTROL?

```
Aaron@HELIOS ~/112_term_project
```

```
$ ls
```

```
termproject_actually_final  termproject_v10  termproject_v3  
termproject_final          termproject_v11  termproject_v4  
termproject_handin         termproject_v12  termproject_v5  
termproject_old_idea       termproject_v13  termproject_v6  
termproject_superfrogger   termproject_v14  termproject_v7  
termproject_temp           termproject_v15  termproject_v8  
termproject_this_one_works termproject_v16  termproject_v9  
termproject_v1             termproject_v2
```


DEVELOPMENT TOOL

When working with a team, the need for a central repository is essential

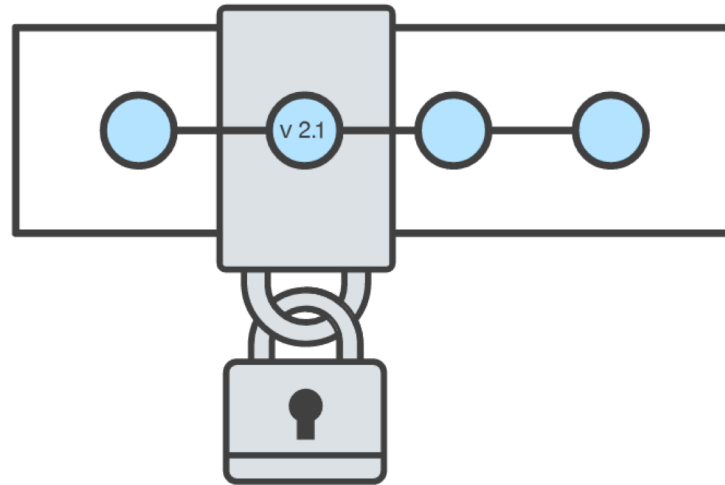
- Need a system to allow versioning, and a way to acquire the latest edition of the code
- A system to track and manage bugs was also needed

GOALS OF VERSION CONTROL

Be able to search through revision history and retrieve previous versions of any file in a project

Be able to share changes with collaborators on a project

Be able to confidently make large changes to existing files



NAMED FOLDERS APPROACH

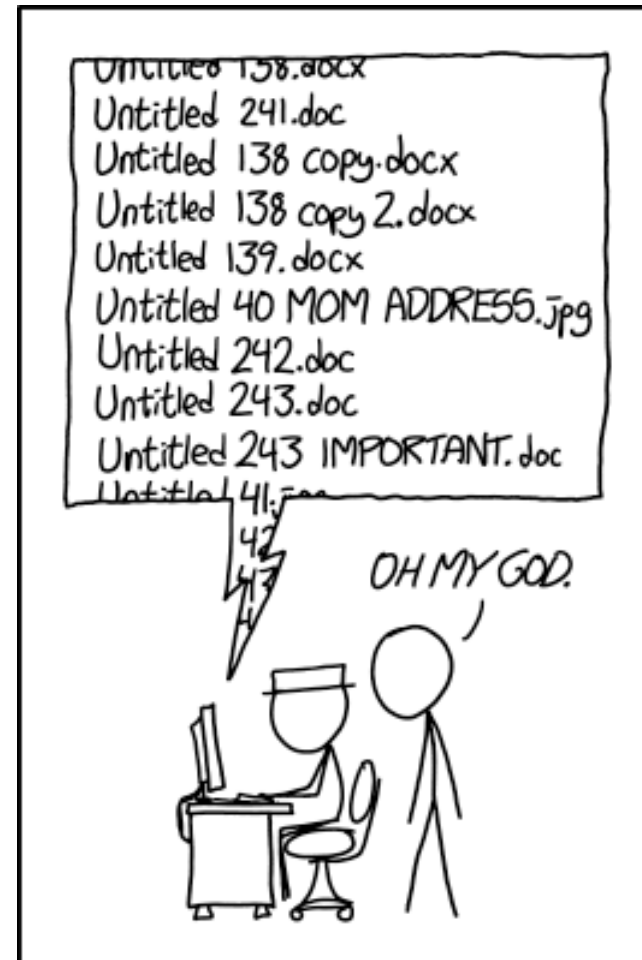
Can be hard to track

Memory-intensive

Can be slow

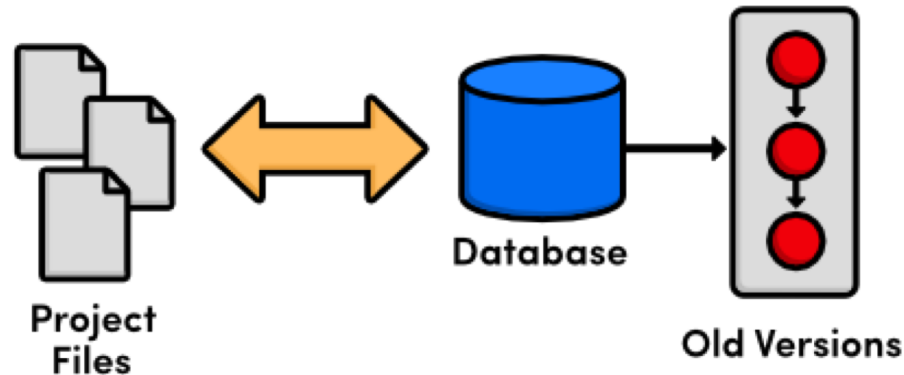
Hard to share

No record of authorship



PRO TIP: NEVER LOOK IN SOMEONE
ELSE'S DOCUMENTS FOLDER.

LOCAL DATABASE OF VERSIONS APPROACH



Provides an abstraction over finding the right versions of files and replacing them in the project

Records who changes what, but hard to parse that

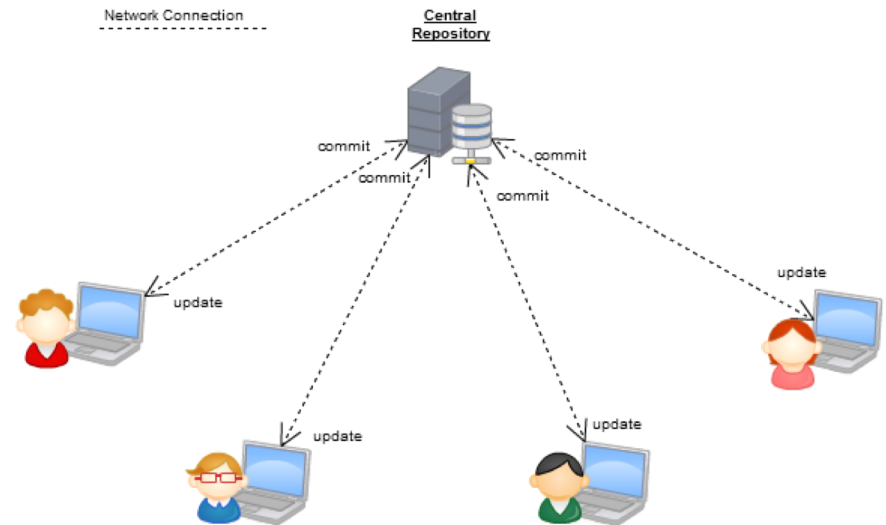
Can't share with collaborators

CENTRALIZED VERSION CONTROL SYSTEMS

A central, trusted repository determines the order of commits (“versions” of the project)

Collaborators “push” changes (commits) to this repository.

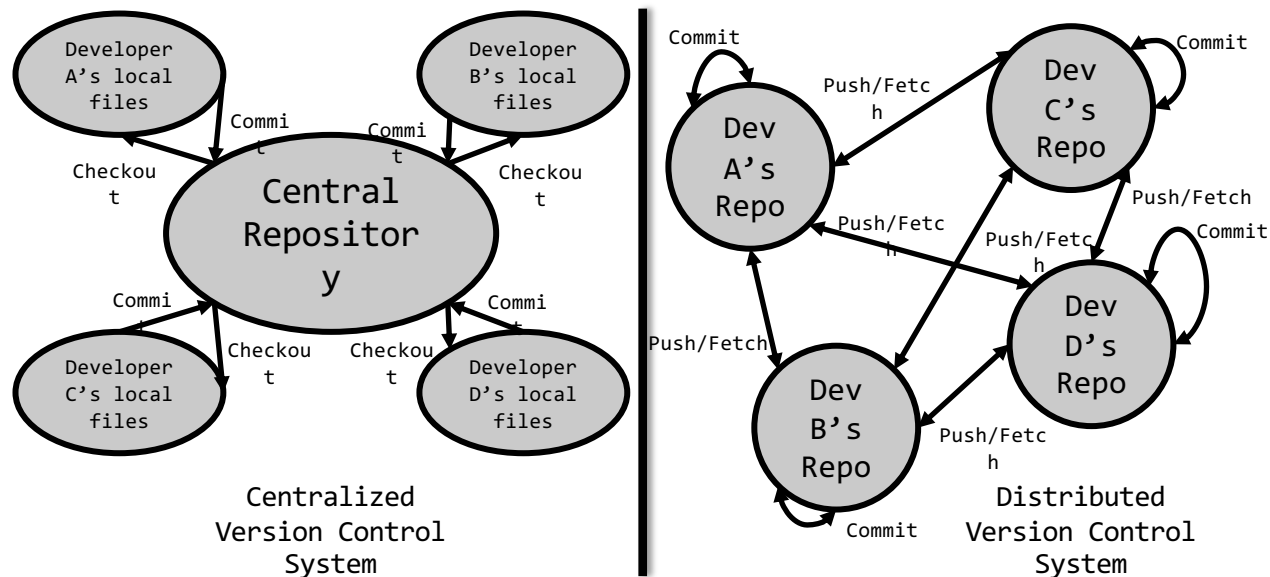
Any new commits must be compatible with the most recent commit. If it isn’t, somebody must “merge” it in.



Examples: SVN, CVS, Perforce

DISTRIBUTED VERSION CONTROL SYSTEMS (DVCS)

- No central repository
- Every repository has every commit
- Examples: **Git**, Mercurial



WHAT IS GIT

Git is a version control system

Developed as a repository system for both local and remote changes

Allows teammates to work simultaneously on a project

Tracks each commit, allowing for a detailed documentation of the project along every step

Allows for advanced merging and branching operations



A SHORT HISTORY OF GIT

Linux kernel development

1991-2002

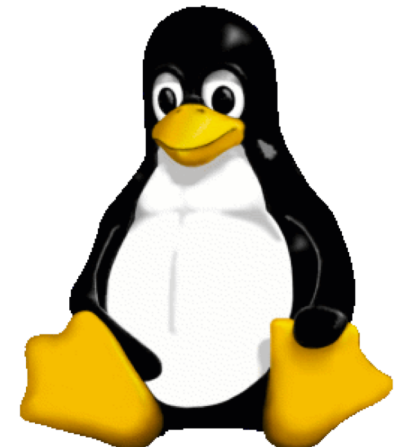
- Changes passed around as archived file

2002-2005

- Using a DVCS called BitKeeper

2005

- Relationship broke down between two communities (BitKeeper licensing issues)



A SHORT HISTORY OF GIT

Goals:

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- **Fully distributed** – not a requirement, can be centralized
- Able to handle large projects like the Linux kernel efficiently (speed and data size)

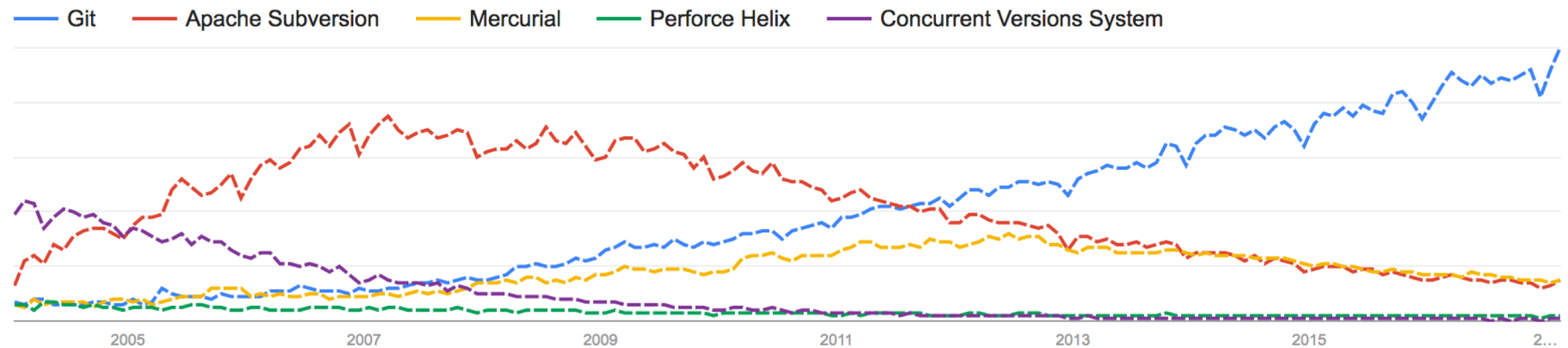
A SHORT HISTORY OF GIT

Popularity:

- Git is now the most widely used source code management tool
- 33.3% of professional software developers use Git (often through GitHub) as their primary source control system

[citation needed]

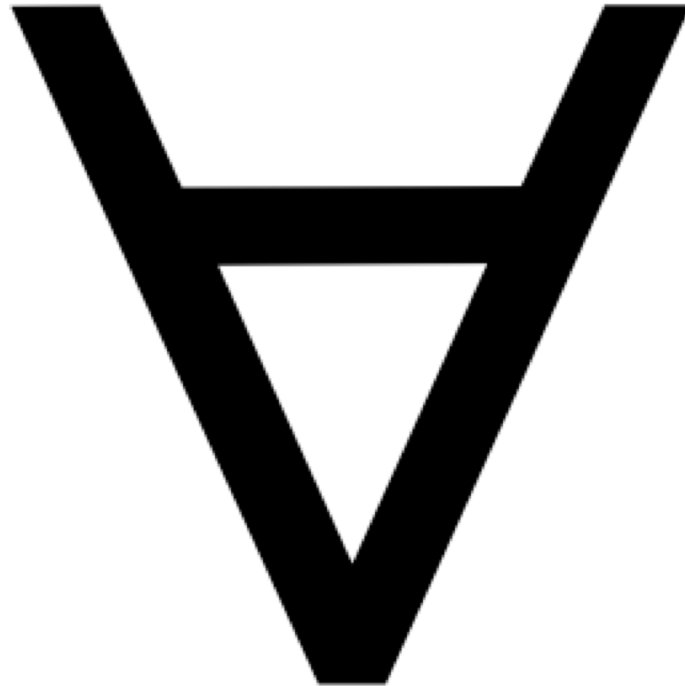
Interest over time. Web Search. Worldwide, 2004 - present.



GIT IN INDUSTRY

Companies and projects currently using Git

- Google
- Android
- Facebook
- Microsoft
- Netflix
- Linux
- Ruby on Rails
- Gnome
- KDE
- Eclipse
- X.org



GIT BASICS

Snapshots, not changes

- A picture of what all your files look like at that moment
- If a file has not changed, store a reference

Nearly every operation is local

- Browsing the history of project
- See changes between two versions

WHY GIT IS BETTER

Git tracks the content rather than the files

Branches are lightweight, and merging is a simple process

Allows for a more streamlined offline development process

Repositories are smaller in size and are stored in a single .git directory

Allows for advanced staging operations, and the use of stashing when working through troublesome sections

WHAT ABOUT SVN?

Subversion has been the most pointless project ever started ... Subversion used to say CVS done right: with that slogan there is nowhere you can go. There is no way to do CVS right ... If you like using CVS, you should be in some kind of mental institution or somewhere else.



Linus Torvalds

GIT VS {CVS, SVN, ...}

Why you should care:

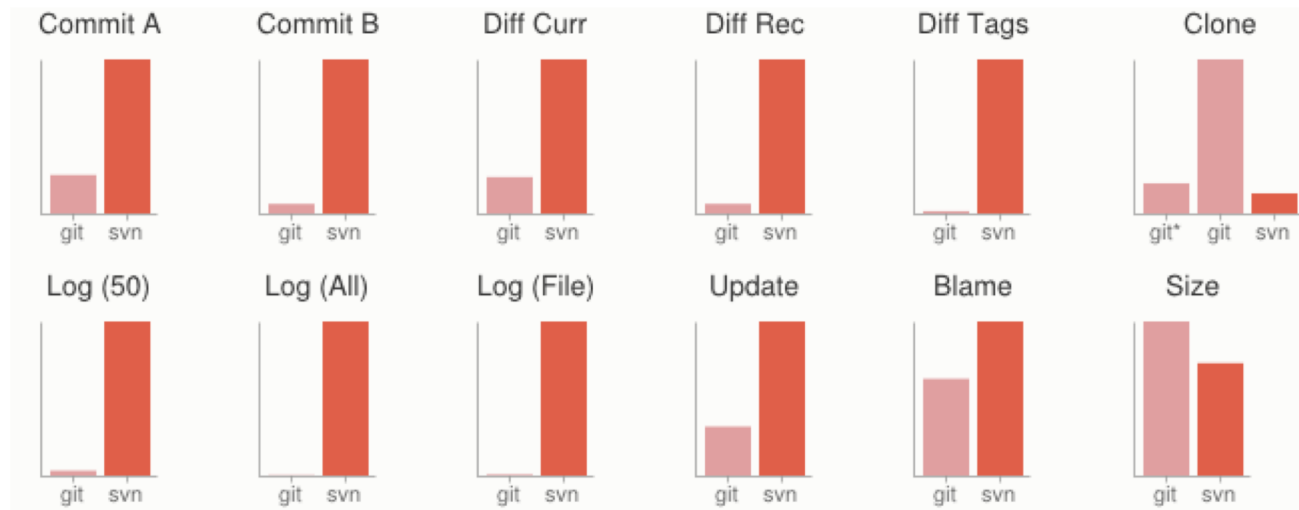
- Many places use legacy systems that will cause problems in the future – be the change you believe in!

Git is **much** faster than SVN:

- Coded in C, which allows for a great amount of optimization
- Accomplishes much of the logic client side, thereby reducing time needed for communication
- Developed to work on the Linux kernel, so that large project manipulation is at the forefront of the benchmarks

GIT VS {CVS, SVN, ...}

Speed benchmarks:



Benchmarks performed by <http://git-scm.com/about/small-and-fast>

GIT VS {CVS, SVN, ...}

Git is significantly smaller than SVN

- All files are contained in a small decentralized .git file
- In the case of Mozilla's projects, a Git repository was 30 times smaller than an identical SVN repository
- Entire Linux kernel with 5 years of versioning contained in a single 1 GB .git file
- SVN carries two complete copies of each file, while Git maintains a simple and separate 100 bytes of data per file, noting changes and supporting operations

Nice because you can (and do!) store the whole thing locally



GIT VS {CVS, SVN, ...}

Git is more **secure** than SVN

- All commits are uniquely hashed for both security and indexing purposes
- Commits can be authenticated through numerous means
 - In the case of SSH commits, a key may be provided by both the client and server to guarantee authenticity and prevent against unauthorized access

GIT VS {CVS, SVN, ...}

Git is decentralized:

- Each user contains an individual repository and can check commits against itself, allowing for detailed local revisioning
- Being decentralized allows for easy replication and deployment
- In this case, SVN relies on a single centralized repository and is unusable without

GIT VS {CVS, SVN, ...}

Git is **flexible**:

- Due to its decentralized nature, git commits can be stored locally, or committed through HTTP, SSH, FTP, or even by Email
- No need for a centralized repository
- Developed as a command line utility, which allows a large amount of features to be built and customized on top of it

GIT VS {CVS, SVN, ...}

Data assurance: a checksum is performed on both upload and download to ensure sure that the file hasn't been corrupted.

Commit IDs are generated upon each commit:

- Linked list style of commits
- Each commit is linked to the next, so that if something in the history was changed, each following commit will be rebranded to indicate the modification

GIT VS {CVS, SVN, ...}

Branching:

- Git allows the usage of advanced **branching** mechanisms and procedures
- Individual divisions of the code can be separated and developed separately within separate branches of the code
- Branches can allow for the separation of work between developers, or even for disposable experimentation
- Branching is a precursor and a component of the merging process

Will give an example shortly.

GIT VS {CVS, SVN, ...}

Merging

- The process of merging is directly related to the process of branching
- Individual branches may be merged together, solving code conflicts, back into the default or master branch of the project
- Merges are usually done automatically, unless a conflict is presented, in which case the user is presented with several options with which to handle the conflict

Will give an example shortly.

GIT VS {CVS, SVN, ...}

Merging: content of the files is tracked rather than the file itself:

- This allows for a greater element of tracking and a smarter and more automated process of merging
- SVN is unable to accomplish this, and will throw a conflict if, e.g., a file name is changed and differs from the name in the central repository
- Git is able to solve this problem with its use of managing a local repository and tracking individual changes to the code

INITIALIZATION OF A GIT REPOSITORY

```
C:\> mkdir CoolProject
C:\> cd CoolProject
C:\CoolProject > git init
Initialized empty Git repository in
C:/CoolProject/.git
C:\CoolProject > notepad README.txt
C:\CoolProject > git add .
C:\CoolProject > git commit -m 'my first
commit'
[master (root-commit) 7106a52] my first commit
1 file changed, 1 insertion(+)
create mode 100644 README.txt
```



GIT BASICS I

The three (or four) states of a **file**:

- **Modified:**
 - File has changed but not committed
- **Staged:**
 - Marked to go to next commit snapshot
- **Committed:**
 - Safely stored in local database
- **Untracked!**
 - Newly added or removed files

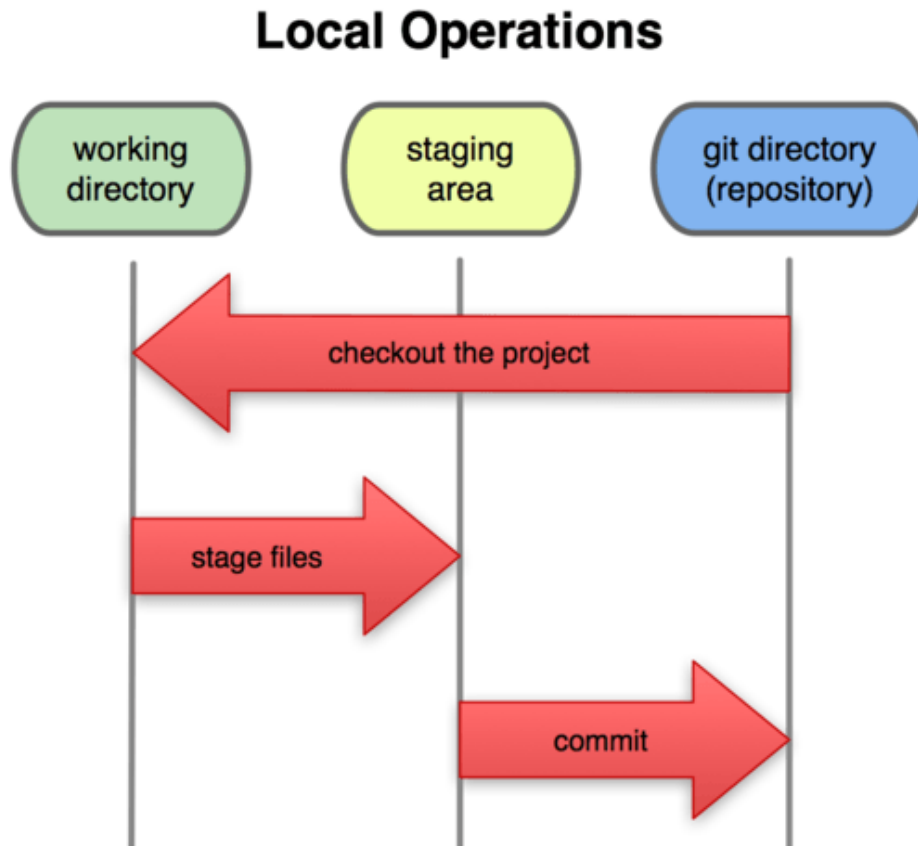
GIT BASICS II

Three main areas of a git **project**:

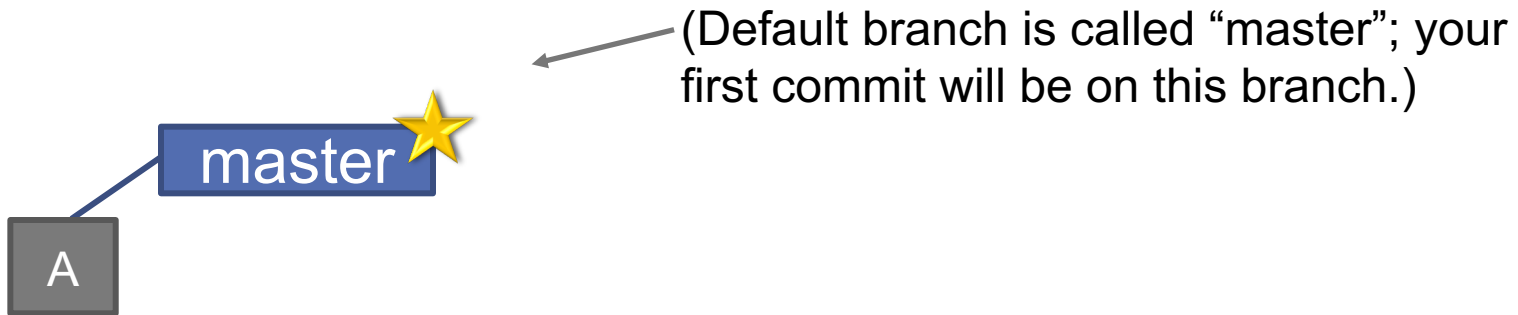
- **Working directory**
 - Single checkout of one version of the project.
- **Staging area**
 - Simple file storing information about what will go into your next commit
- **Git directory**
 - What is copied when cloning a repository

GIT BASICS III

Three main areas of a git **project**:

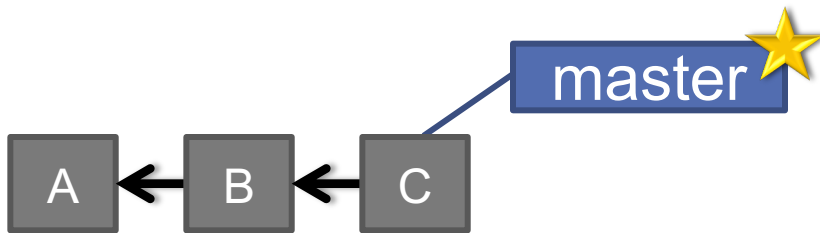


BRANCHES ILLUSTRATED



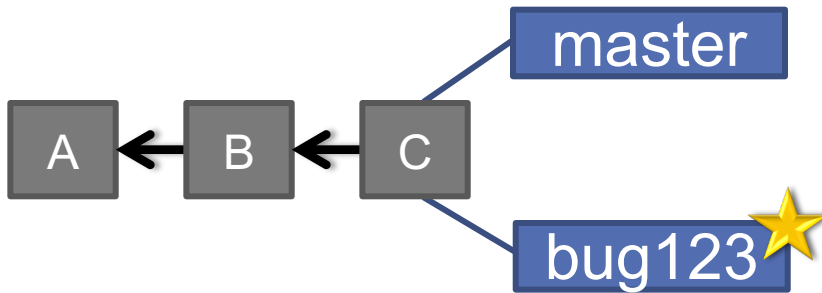
```
> git commit -m 'my first commit'
```

BRANCHES ILLUSTRATED



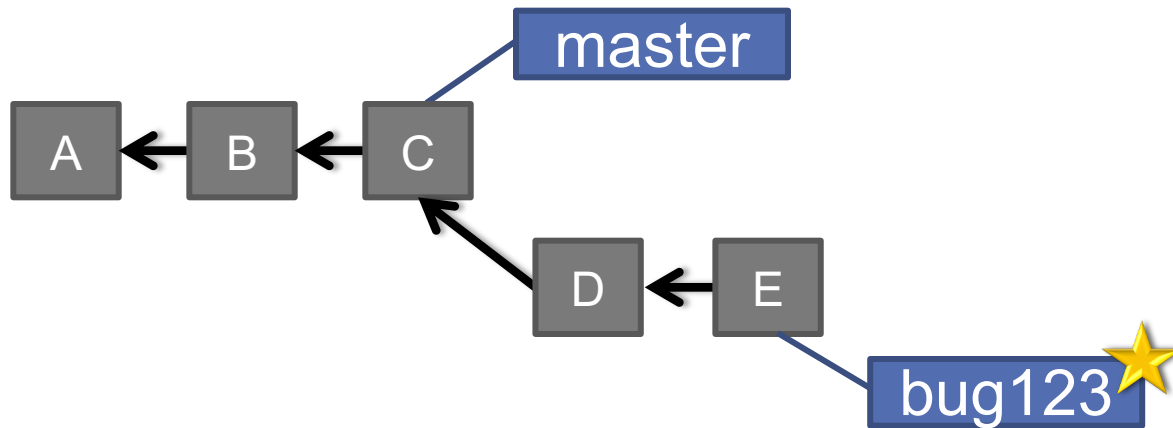
```
> git commit (x2)
```

BRANCHES ILLUSTRATED



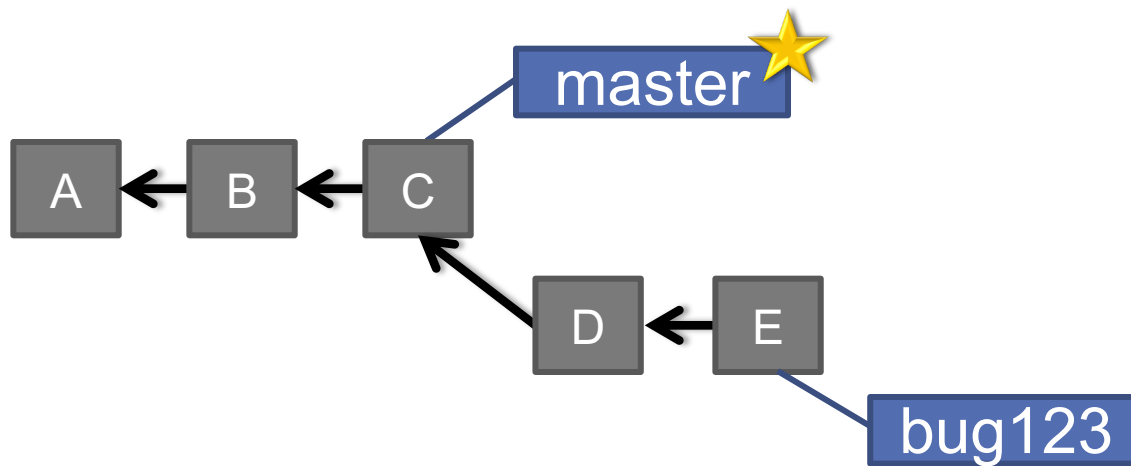
```
> git checkout -b bug123
```

BRANCHES ILLUSTRATED



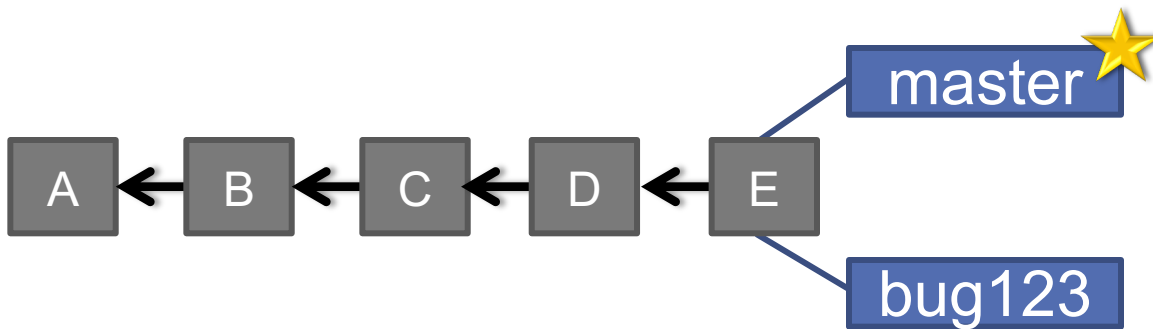
```
> git commit (x2)
```


BRANCHES ILLUSTRATED



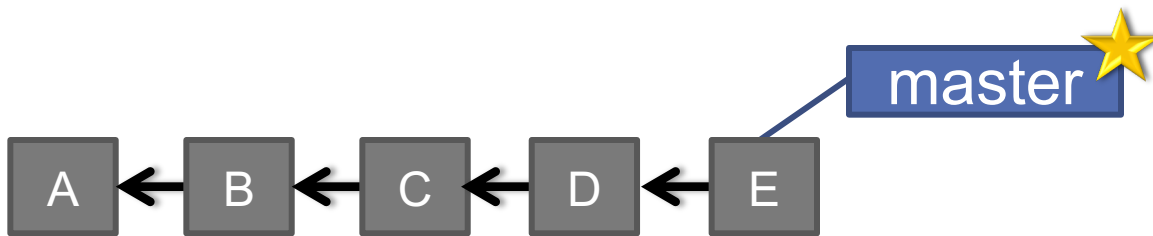
```
> git checkout master
```

BRANCHES ILLUSTRATED



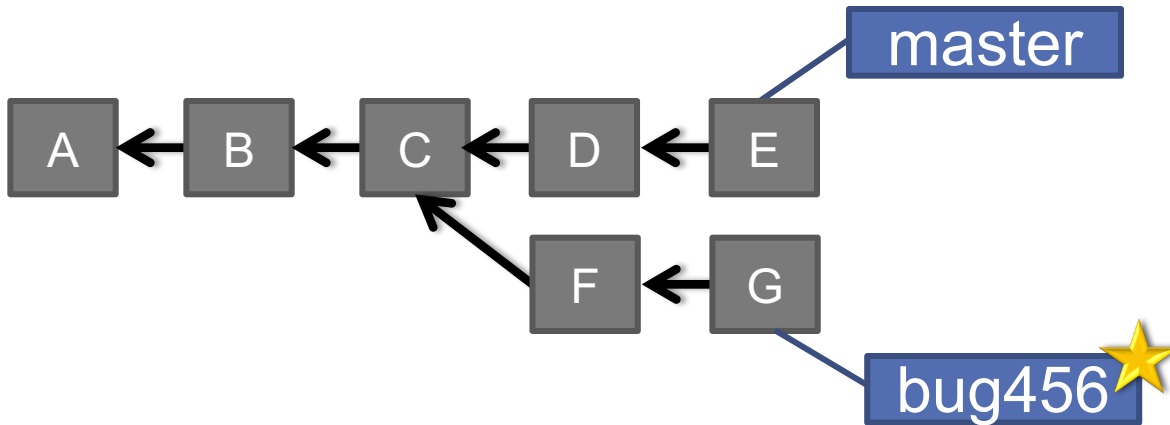
```
> git merge bug123
```

BRANCHES ILLUSTRATED

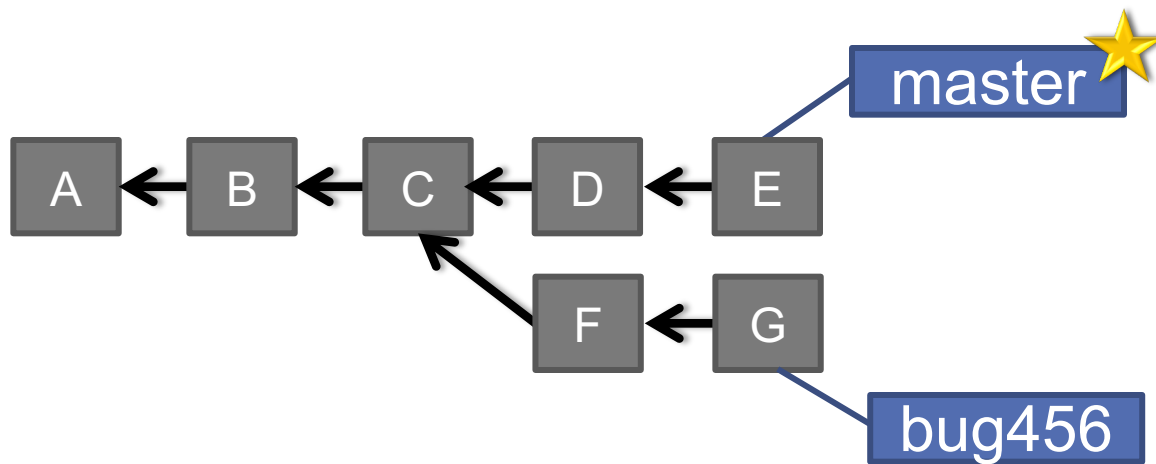


```
> git branch -d bug123
```

BRANCHES ILLUSTRATED

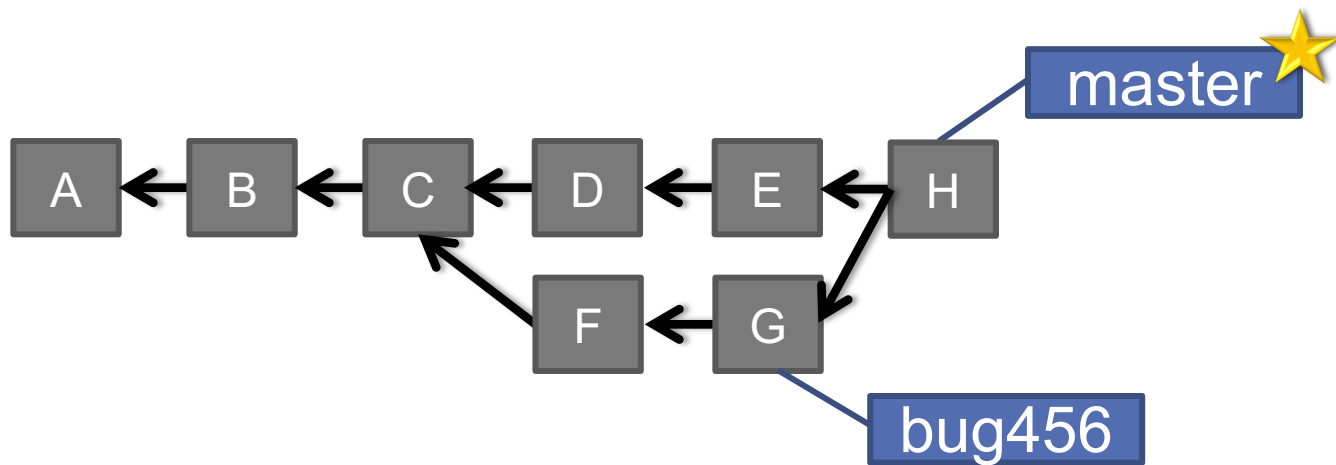


BRANCHES ILLUSTRATED



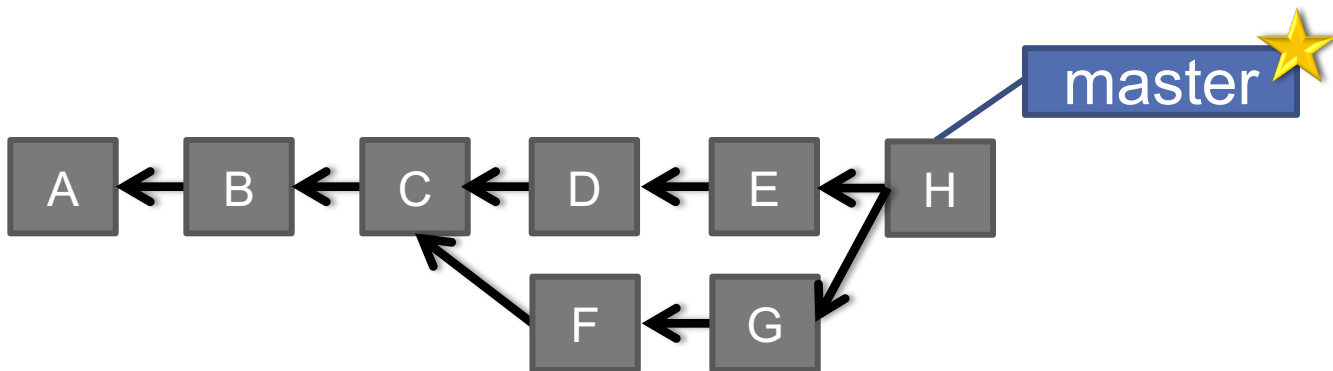
```
> git checkout master
```

BRANCHES ILLUSTRATED



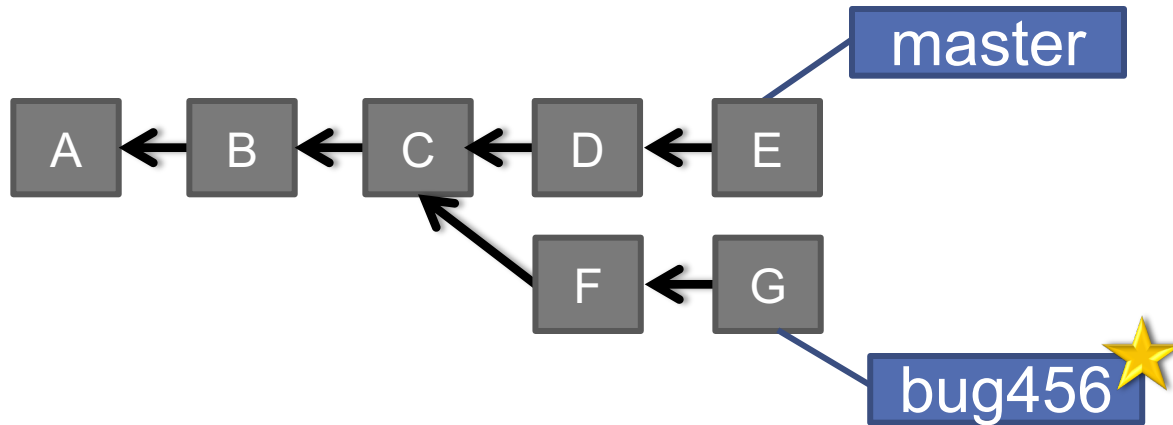
```
> git merge bug456
```

BRANCHES ILLUSTRATED

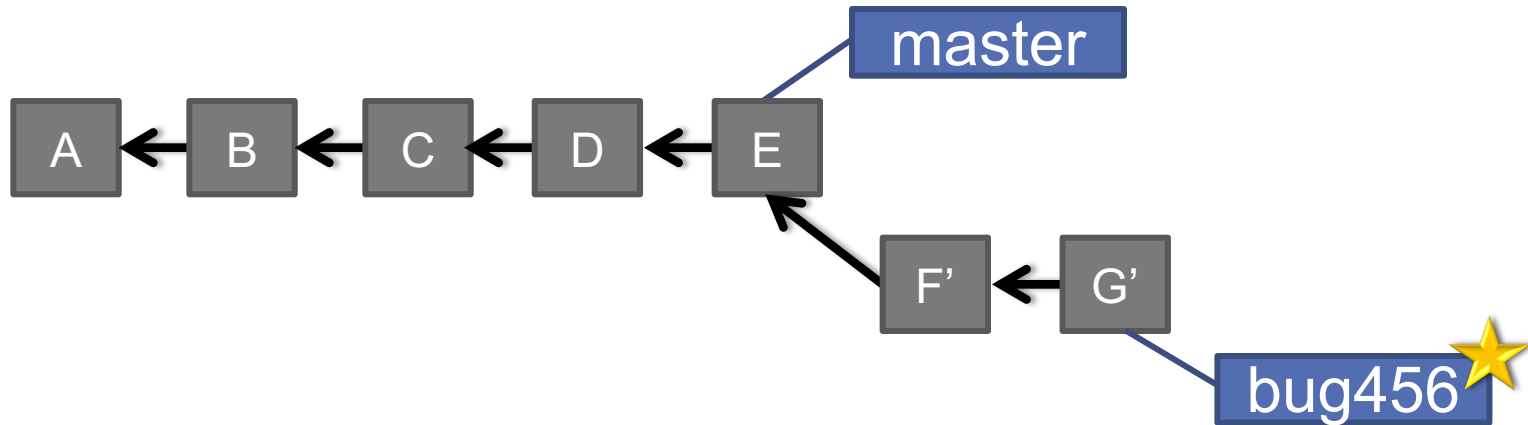


```
> git branch -d bug456
```

BRANCHES ILLUSTRATED

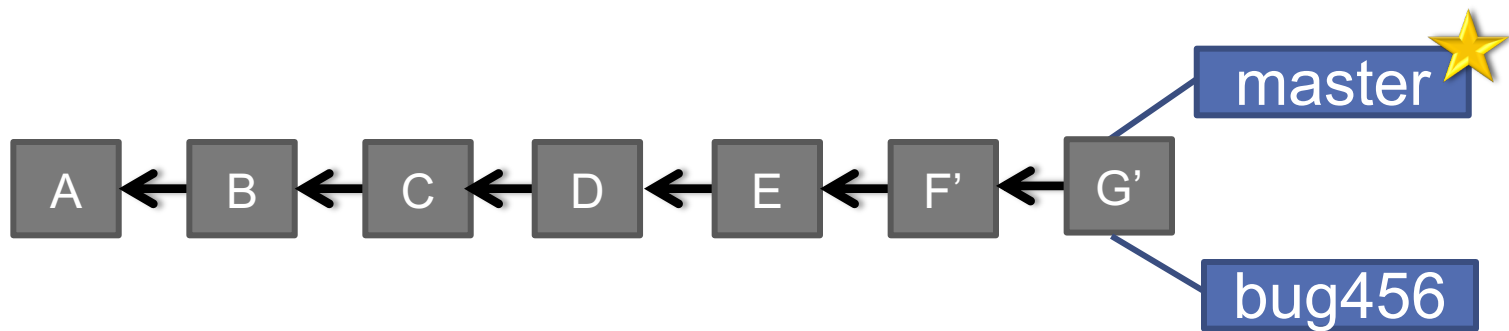


BRANCHES ILLUSTRATED



```
> git rebase master
```

BRANCHES ILLUSTRATED



```
> git checkout master  
> git merge bug456
```

WHEN TO BRANCH?

General rule of thumb:

- **Anything in the master branch is always deployable.**

Local branching is very lightweight!

- New feature? Branch!
- Experiment that you won't ever deploy? Branch!

Good habits:

- Name your branch something descriptive (add-like-button, refactor-jobs, create-ai-singularity)
- Make your commit messages descriptive, too!



SO YOU WANT SOMEBODY ELSE TO HOST THIS FOR YOU ...

Git: general distributed version control system

GitHub / BitBucket / GitLab / ...: **hosting** services for git repositories

In general, GitHub is the most popular:

- Lots of big projects (e.g., Python, Bootstrap, Angular, D3, node, Django, Visual Studio)
- Lots of ridiculously awesome projects (e.g., <https://github.com/maxbbraun/trump2cash>)

There are reasons to use the competitors (e.g., private repositories, access control)



“SOCIAL CODING”



John P. Dickerson
JohnDickerson

Assistant Professor of Computer Science, University of Maryland; Ph.D. in Computer Science, Carnegie Mellon University

👤 University of Maryland
📍 Washington, DC
🌐 <http://jpdickerson.com>

Organizations



[Overview](#) [Repositories 14](#) [Stars 71](#) [Followers 34](#) [Following 47](#)

Popular repositories

Customize your pinned repositories

KidneyExchange
Kidney paired donation optimization code
Java ★ 8 🍴 7

TrackIt
Modular suite that measures a child's sustained selective attention.
Java ★ 3 🍴 2

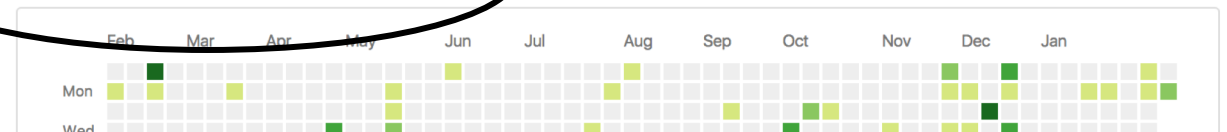
EnvyFree
Computes envy-free allocations of items to agents.
Python ★ 2

VotingRules
Compute winners in elections based on various voting rules.
Java ★ 1

muffins
Fairly feeding hungry students
Python ★ 1

website
My academic website.
TeX

75 contributions in the last year



REVIEW: HOW TO USE

Git commands for everyday usage are relatively simple

- **git pull**
 - Get the latest changes to the code
- **git add .**
 - Add any newly created files to the repository for tracking
- **git add -u**
 - Remove any deleted files from tracking and the repository
- **git commit -m 'Changes'**
 - Make a version of changes you have made
- **git push**
 - Deploy the latest changes to the central repository

Make a repo on GitHub and **clone** it to your machine:

- <https://guides.github.com/activities/hello-world/>

STUFF TO CLICK ON

Git

- <http://git-scm.com/>

GitHub

- <https://github.com/>
- <https://guides.github.com/activities/hello-world/>
- ^-- Just do this one. You'll need it for your tutorial 😊.

GitLab

- <http://gitlab.org/>

Git and SVN Comparison

- <https://git.wiki.kernel.org/index.php/GitSvnComparison>