INTRODUCTION TO DATA SCIENCE

JOHN P DICKERSON

PREM SAGGAR

Today!

Lecture #3 – 9/5/2018

CMSC320
Mondays & Wednesdays
2pm – 3:15pm

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

ANNOUNCEMENTS

Register on Piazza: piazza.com/umd/fall2018/cmsc320

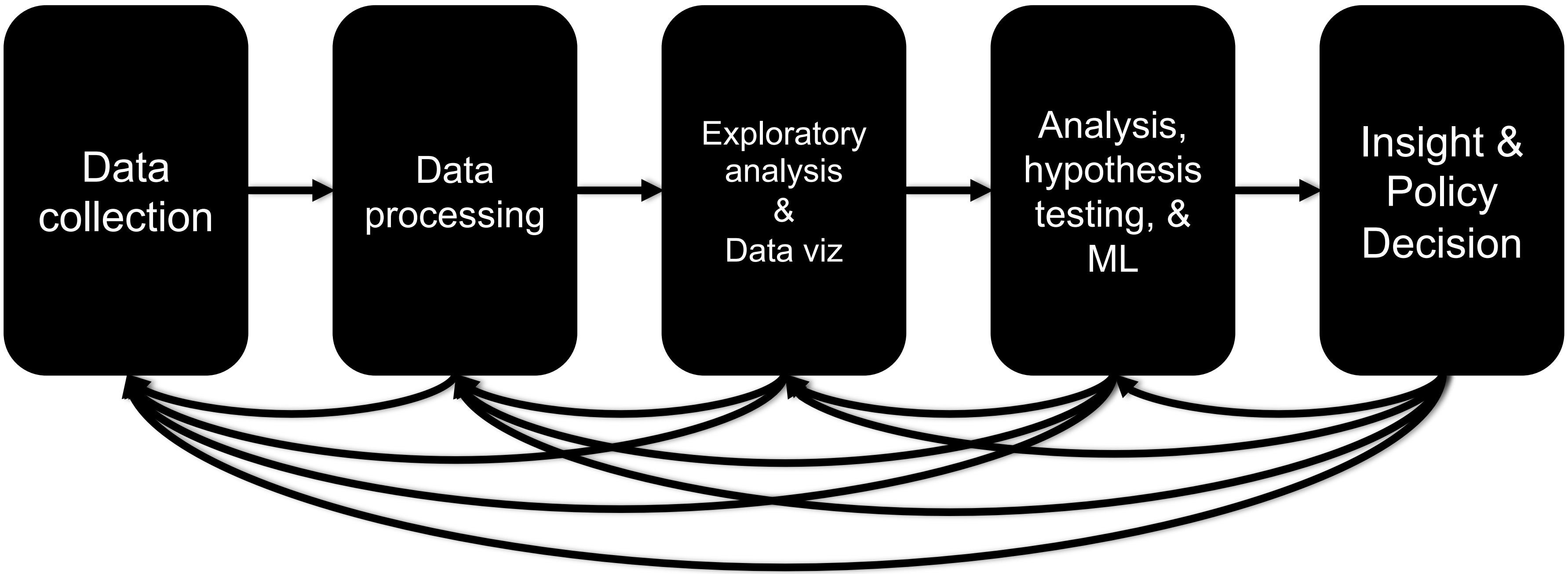
- 219 have registered already ♡
- >6 have not registered yet or not activated account 💔

Office Hours Posted over the Weekend

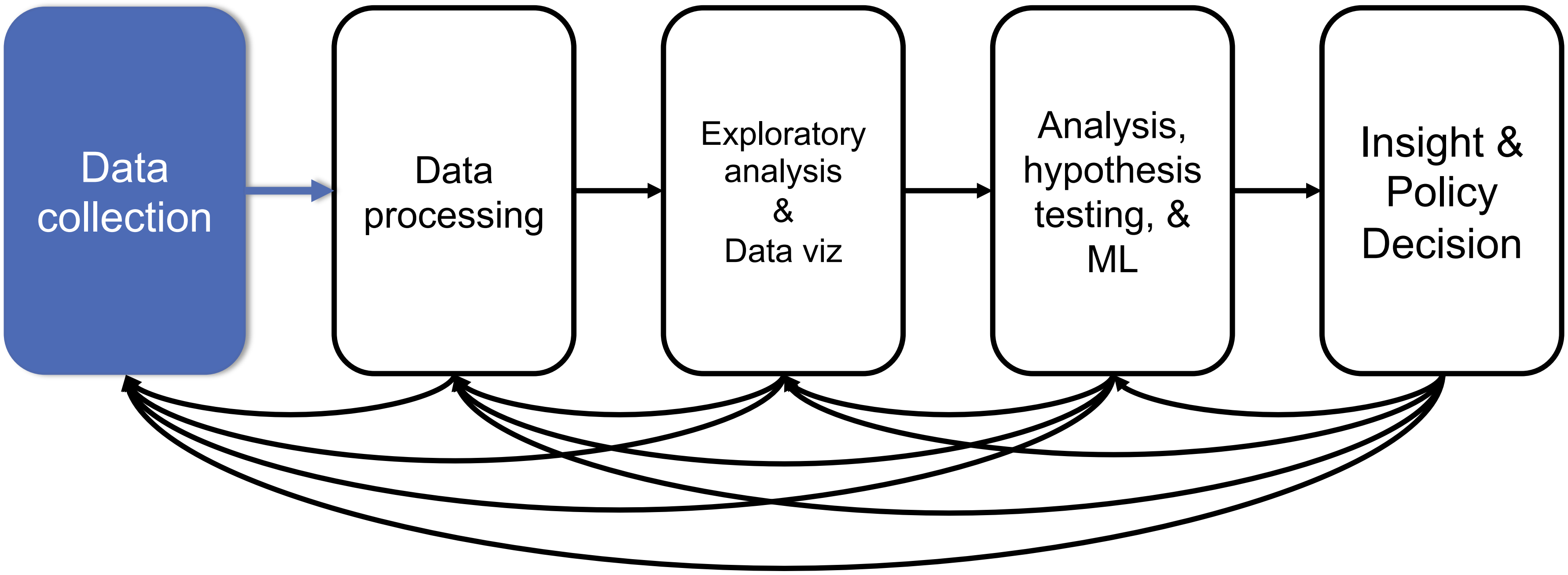
Reminder: Weekly quizzes, due on Mondays at noon (except first one that was due today)

Project 1 will be released soon (mid-next week)

THE DATA LIFECYCLE



WRAPPING UP LAST LECTURE ....



WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

REGULAR EXPRESSIONS

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

Indispensable for data munging and wrangling

Many constructs to search a variety of different patterns

Many languages/libraries (including Python) allow “compiling”

- Much faster for repeated applications of the regex pattern
- https://blog.codinghorror.com/to-compile-or-not-to-compile/

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')
```

SIMPLE EXAMPLE: PARSE AN EMAIL ADDRESS

```
(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t] )+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:
\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:( ?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[
\t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\0 31]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\
](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+ (?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:
(?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z |(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n) ?[
\t])*)*\<(?:(?:\r\n)?[ \t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\ r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[
\t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n) ?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t]
)*))*(?:,@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*
)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t] )+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*)
*:(?:(?:\r\n)?[ \t])*)?(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+ |\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r
\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?: \r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t
]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031 ]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](
?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(? :(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?
:\r\n)?[ \t])*))*\>(?:(?:\r\n)?[ \t])*)|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(? :(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)? [
\t]))*"(?:(?:\r\n)?[ \t])*)*:(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|
\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<> @,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"
(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t] )*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\
".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(? :[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[
\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\\".\[\] \000- \031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(
?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)*\<(?:(?:\r\n)?[ \t])*(?:@(?:[^()<>@,; :\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([
^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\" .\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\
]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*(?:,@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\ [\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\
r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]
|\\.)*\](?:(?:\r\n)?[ \t])*))*)*:(?:(?:\r\n)?[ \t])*)?(?:[^()<>@,;:\\".\[\] \0 00-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\
.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@, ;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?
:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])* (?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".
\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[ ^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]
]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*\>(?:(?:\r\n)?[ \t])*)(?:,\s*( ?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\
".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:( ?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[
\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t ])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t
])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(? :\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|
\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*|(?: [^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\
]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)*\<(?:(?:\r\n) ?[ \t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["
()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n) ?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>
@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*(?:,@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,
;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t] )*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\
".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*)*:(?:(?:\r\n)?[ \t])*)? (?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".
\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?: \r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\[
"()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t]) *))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])
+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\ .(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z
|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*\>(?:( ?:\r\n)?[ \t])*))*)?;\s*)
```

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 Introduction to Data Science"
re.sub(r"Data Science", r"Schmada Schmience", text)
```

```
'I love Introduction to Schmada Schmience`
```

Can incorporate groups into the matching

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

Thanks to: Zico Kolter

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

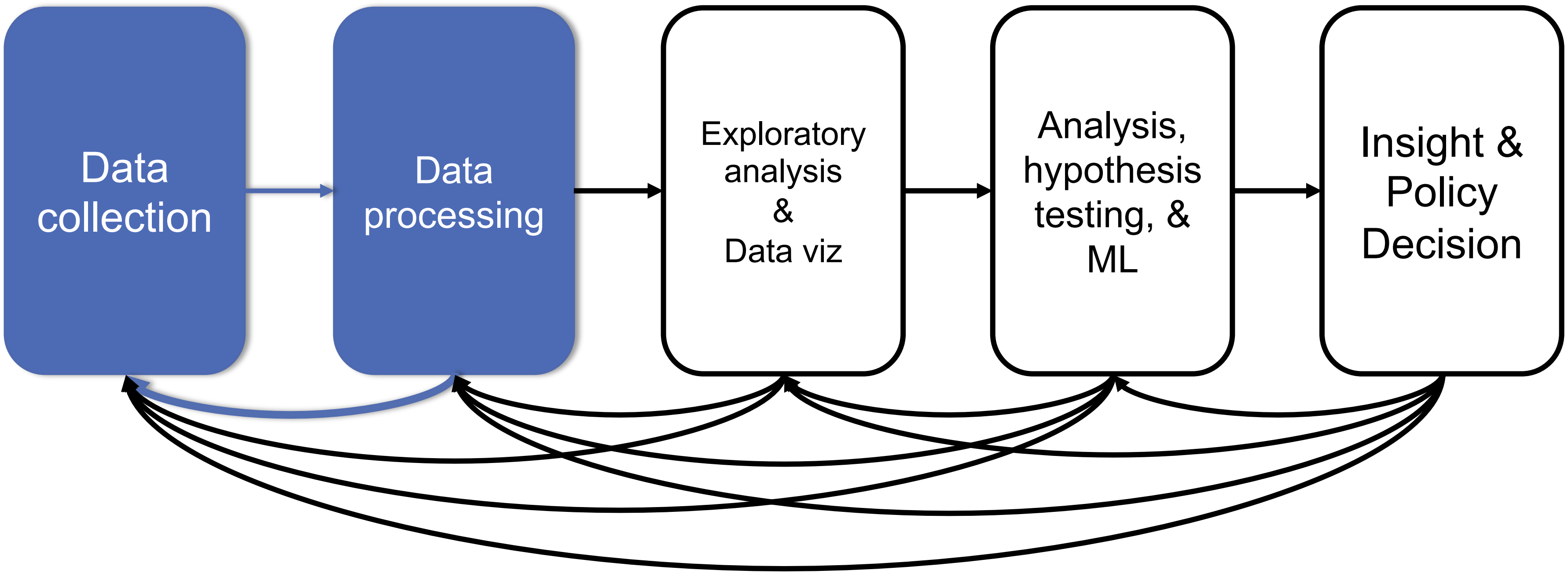
Parse exactly what you want

```
# 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)
```

TODAY'S LECTURE



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 420, 424), 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)

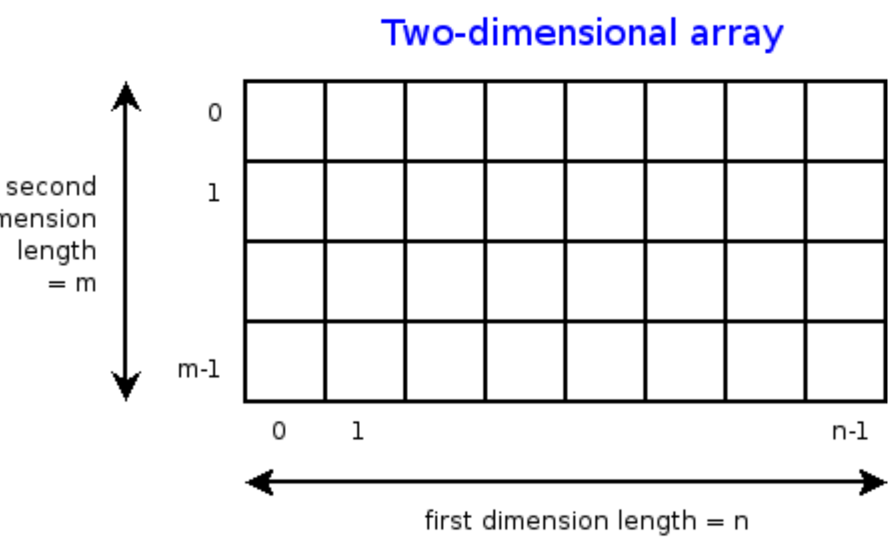
Given two vectors: **Dot and cross products**

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



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

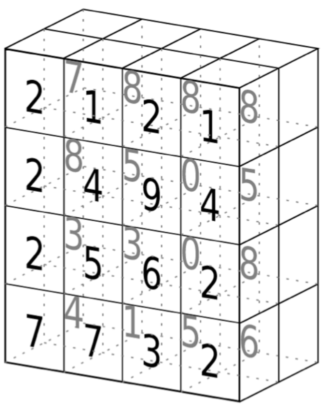
DATA MANIPULATION AND COMPUTATION

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]

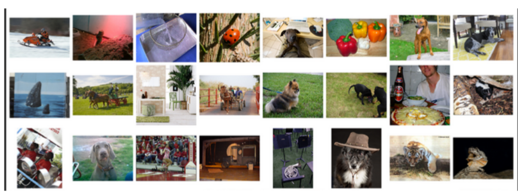
n-dimensional array operations
+
Linear Algebra
- **Matrix/tensor multiplication**
- **Transpose**
- **Matrix-vector multiplication**
- **Matrix factorization**

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



Sets: of (Key, Value Pairs)

(amol@cs.umd.edu,(email1, email2,…))

(john@cs.umd.edu,(email3, email4,…))

Filter
Map
Union

Reduce/Aggregate

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)

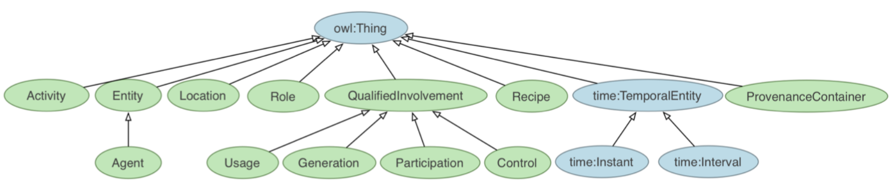
Many other languages and constructs, that look very similar

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 FEW 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 FEW 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.

Many, many thanks to: FSU CIS4930

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

[FSU]

THE NUMPY STACK

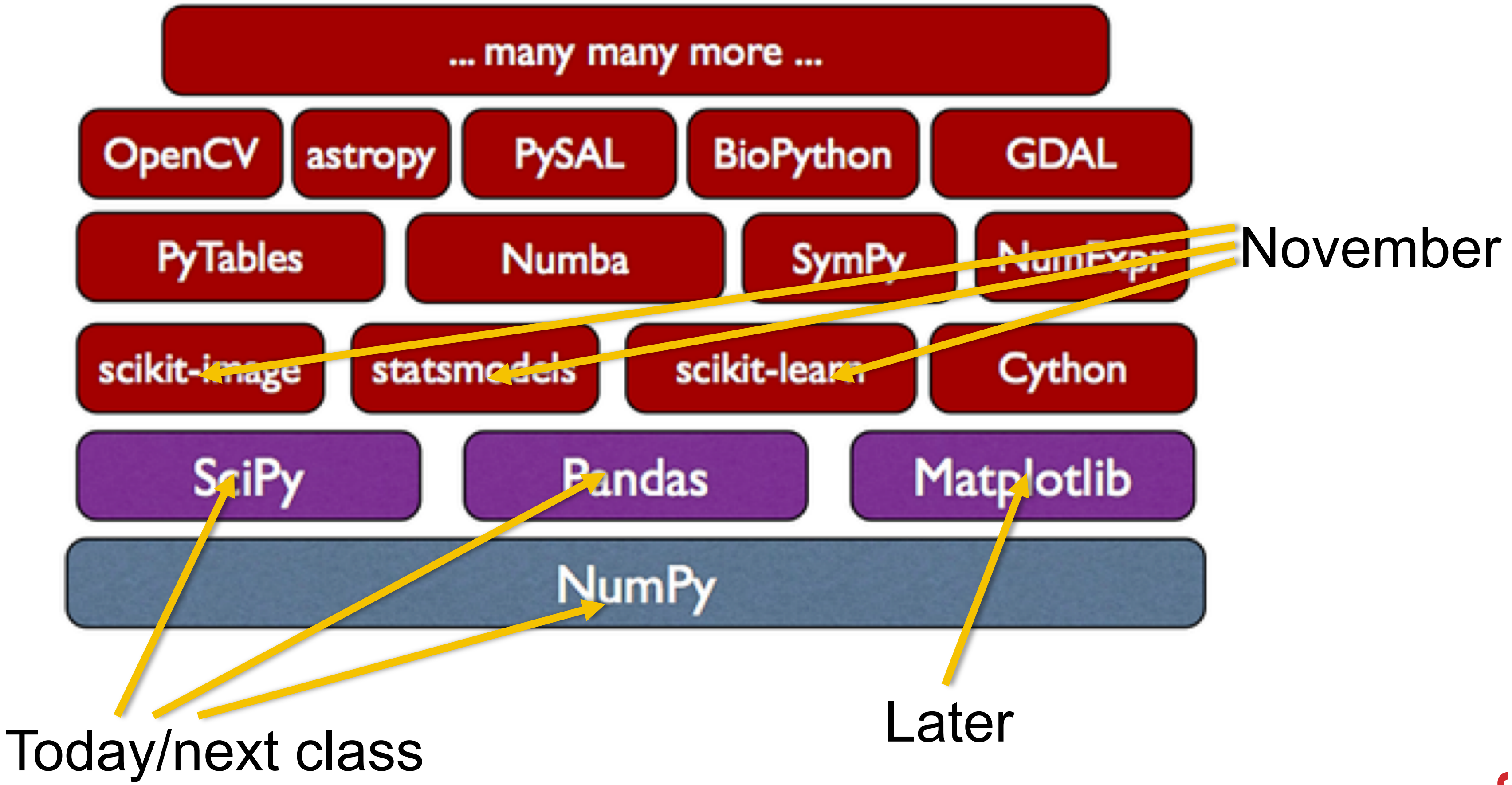


Image from Continuum Analytics

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.

[FSU]

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.

[FSU]

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.

[FSU]

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')
```

[FSU]

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)

[FSU]

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]])
```

[FSU]

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])
```

[FSU]

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]])
```


[FSU]

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]]]
```


[FSU]

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

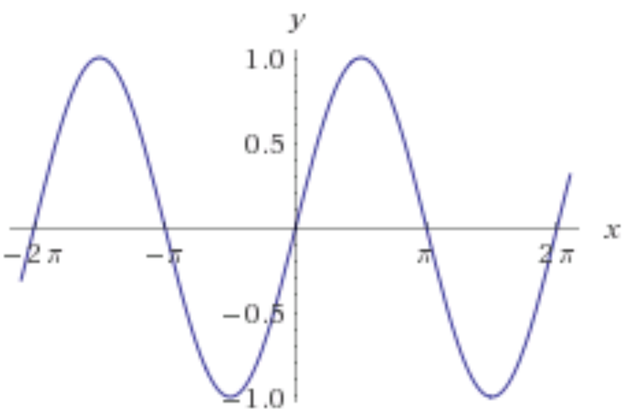
ONE SCIPY EXAMPLE

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 tomorrow