UNIVERSITY OF MARYLAND

INST447 -0101 Fall 2020 Lecture 6

Virtual

Instructor: Bill Farmer

TA: Jonathan Chen

Grader: Jeffrey Chen

October 6 & 8, 2020

This Week

Time: Tuesday virtual

- Admin
  - Syllabus Updates
  - Grades
  - Midterm
- Metadata
- XML
- Json
- SQL -Sqlite

Time: Thursday Virtual
w/ optional live session

- Live session
  - Jupyter Notebooks subjects from reading
- Videos
  - Data structures
- Lab & Midterm & Review (Next week)
- Projects - teams

If you are tired, stand up in the back of class.

Use of phone during class for non-class purposes is rude.

COLLEGE OF INFORMATION STUDIES

Admin

Admin

- Office Hours (need to schedule a time slot):
  - Monday 8-9 pm
  - Friday 8-10 am
  - Saturday 6-8 pm (changed from am to pm)
  - Sunday 6-7 pm (changed from 4-6 to 6-7)
  - By Appointment * Anytime
- Live class meetings - Thursdays 12:30-1:30
  - Class originally scheduled to start @ 12:30 so I figure this is a good time
  - We can add a couple of these at different times if/when needed
- Midterm Next week and Review Session

INST447 General Schedule

- Update 9/15/2020

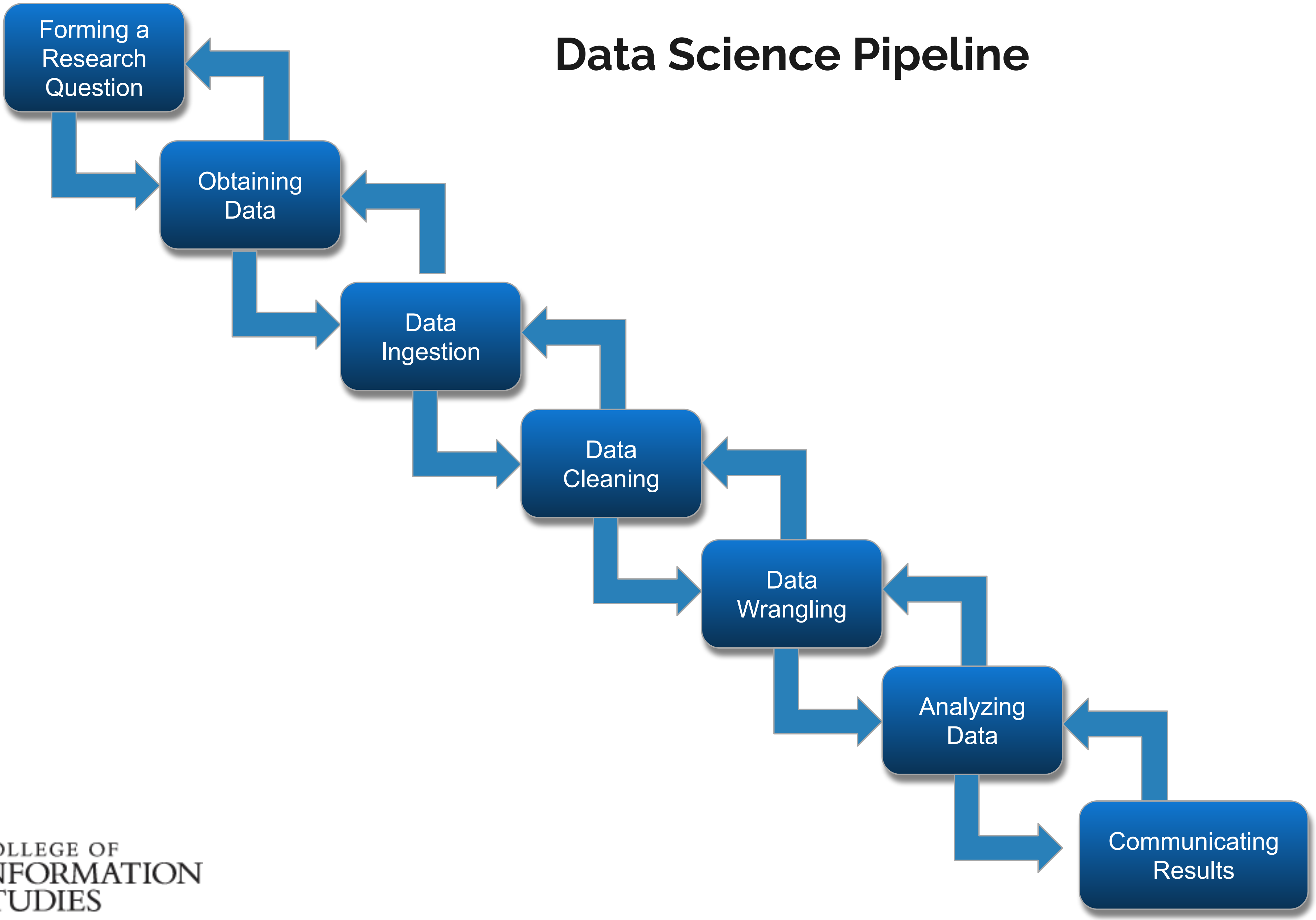
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
					🟠 8-10am	
		🔶 Noon-ish		▲ 12:30-1:30pm		
🟠 6-7pm	🔵 8-9 pm					
	▰ Lab 11:59pm					🟠 6-8pm

🔵	Office Hours Live
🟠	Office Hours By appointment
🔶	Video Ready
▲	Class Live Sessions
▰	Lab Due

Project Teams

- Project Teams set up in Canvas
- If you have teams, please sign up
- First one to sign up is the team ‘captain’
- Team contract will be due
- Project Proposal due 10/8

Data Science Pipeline



COLLEGE OF INFORMATION STUDIES

Metadata

Metadata

- What is Metadata?
  - Metadata is structured information describing a resource, for example: the dates, title, and creators associated with a dataset.
  - Metadata is "data that provides information about other data". In other words, it is "data about data."
  - More precisely, metadata describes data containing specific information like type, length, textual description and other characteristics.
    - Examples
      - Date of ingest, Date of update
      - Collection method (API, manual, other)
      - Processing servers

https://www.villanovau.com/resources/bi/metadata-importance-in-data-driven-world/
https://www.lib.ncsu.edu/do/data-management/metadata

Types of Metadata

Metadata falls into **three** main categories:

1. Descriptive: or used for discovery and identification and including such information as title, author, abstract and keywords.
2. Structural: which shows how information is put together – page order to chapters, for example.
    a. **Technical** metadata properties include file types, size, creation date and time, and type of compression. Technical metadata is often used for digital object management and interoperability.
    b. **Preservation** metadata is used in navigation. Example preservation metadata properties include an item's place in a hierarchy or sequence.
3. Administrative: which enables better resource management by showing such information as when and how the resource was created.
    a. **Rights** metadata might include copyright status, rights holder, or license terms.

https://www.villanovau.com/resources/bi/metadata-importance-in-data-driven-world/
https://www.lib.ncsu.edu/do/data-management/metadata
https://www.lifewire.com/metadata-definition-and-examples-1019177
https://guides.ucf.edu/metadata/intrometadata

Purposes of Metadata

- Resource discovery is one of the most common
  - Effective cataloging
    - Identifying resources.
    - Defining them by criteria.
    - Bringing similar resources together.
    - Distinguishing among those that are dissimilar.
- Effective means of organizing electronic resources
  - Typically, links to resources have been organized as lists and built as static webpages, with the names and resources hardcoded in HTML. A more efficient practice, is to use metadata to build these pages. For Web purposes, the information can be extracted and reformatted through use of software tools.
-

https://www.villanovau.com/resources/bi/metadata-importance-in-data-driven-world/
https://www.lib.ncsu.edu/do/data-management/metadata

Purposes of Metadata

- Facilitates interoperability and integration of resources.
  - Using metadata to describe resources enables its understanding by humans as well as machines.
- Facilitates digital identification via standard numbers that uniquely identify the resource the metadata defines.
  - A practice is to combine metadata so that it acts as a set of identifying data that differentiate objects or resources, supporting validation needs.
- Metadata is an important way to protect resources and their future accessibility.
  - Lineage: For archiving and preservation purposes, it takes metadata elements that track the object's lineage, and describe its physical characteristics and behavior so it can be replicated on technologies in the future.

https://www.villanovau.com/resources/bi/metadata-importance-in-data-driven-world/
https://www.lib.ncsu.edu/do/data-management/metadata

Metadata

- Establishing a metadata workflow that sufficiently describes your data is an important part of data management.
- ***** Metadata planning should occur at the beginning of a research project. *****
- **Metadata isn't the data itself!!**

https://www.villanovau.com/resources/bi/metadata-importance-in-data-driven-world/
https://www.lib.ncsu.edu/do/data-management/metadata

Metadata Standards

- There are a number of metadata standards for datasets.
  - Society for American Archivists
    - https://www2.archivists.org/
    - https://www2.archivists.org/sites/all/files/national_library_of_medicine.pdf
    - https://www2.archivists.org/sites/all/files/UniversityofLouisvilleCDMexportHFox.txt
  - Common Standards (many, one example)
    - https://guides.ucf.edu/c.php?g=79118&p=506121
      - Dublin Core : https://www.dublincore.org/specifications/dublin-core/dces/ -
        - The Dublin Core™ Metadata Element Set is a vocabulary of fifteen properties for use in resource description. The name "Dublin" is due to its origin at a 1995 invitational workshop in Dublin, Ohio; "core" because its elements are broad and generic, usable for describing a wide range of resources.

https://www.villanovau.com/resources/bi/metadata-importance-in-data-driven-world/
https://www.lib.ncsu.edu/do/data-management/metadata
https://www.lifewire.com/annot-file-2619632

Metadata Examples

- **Metadata and Website Searches**
  - The metadata embedded in websites is critically important to the success of the site. It includes a description of the website, keywords, metatags, and more.
  - Some common metadata terms used when building a web page include meta title and meta description.
    - The meta title briefly explains the topic of the page to help readers understand what they'll get from the page should they open it.
    - The meta description is further information, though brief, about the contents of the page.
    - Both of these metadata pieces are displayed on search engines
    - A web page's metadata might also include the language the page was written in, like whether it's an HTML page or the actual language.
- **Metadata for Tracking**
  - Retailers and online shopping sites use metadata to track consumers' habits and movements.
  - ISPs, governments, and anyone else with access to large collections of metadata information could potentially use the metadata from web pages, emails, and other places there are users online, to monitor web activity.

https://www.villanovau.com/resources/bi/metadata-importance-in-data-driven-world/
https://www.lib.ncsu.edu/do/data-management/metadata
https://www.lifewire.com/annot-file-2619632

Metadata Examples

- **Metadata in Social Media**
  - Geotagged images
  - You can see a profile image and a short description of the Facebook user to learn just the basics about them before deciding to friend them or send them a message

https://www.villanovau.com/resources/bi/metadata-importance-in-data-driven-world/
https://www.lib.ncsu.edu/do/data-management/metadata
https://www.lifewire.com/annot-file-2619632

XML

XML

- **eXtensible Markup Language**
  - One of the two common formats that we use when exchanging data across the web. (JSON)
  - XML looks very similar to HTML, but XML is more structured than HTML. Here is a sample of an XML document.
  - It was designed to store and transport data. It was designed to be both human- and machine-readable.
- Each pair of opening (e.g., `<person>`) and closing tags (e.g., `</person>`) represents a ***element*** or ***node*** with the same name as the tag (e.g., `person`).
- Each element can have:
  - Text
  - Attributes (e.g., `hide`)
  - Other nested elements.

- If an XML element is empty (i.e., has no content), then it may be depicted by a self-closing tag (e.g., `<email />`).

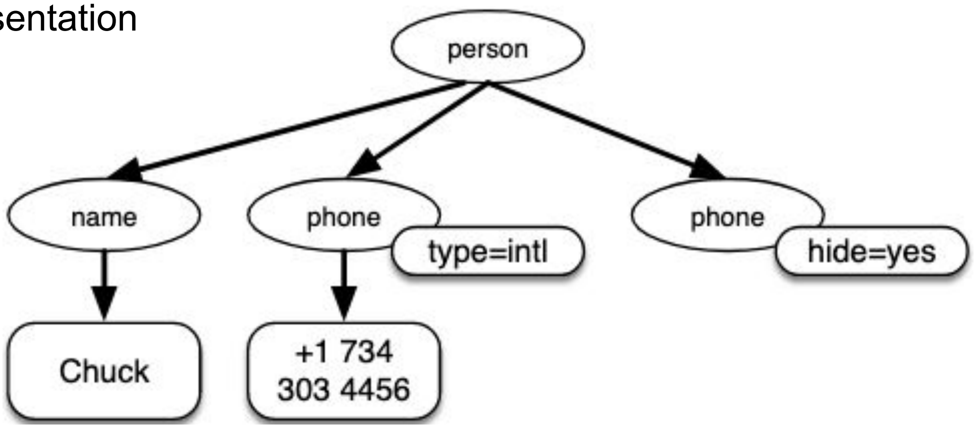
```
<person>
  <name>Chuck</name>
  <phone type="intl">
    +1 734 303 4456
  </phone>
  <email hide="yes" />
</person>
```

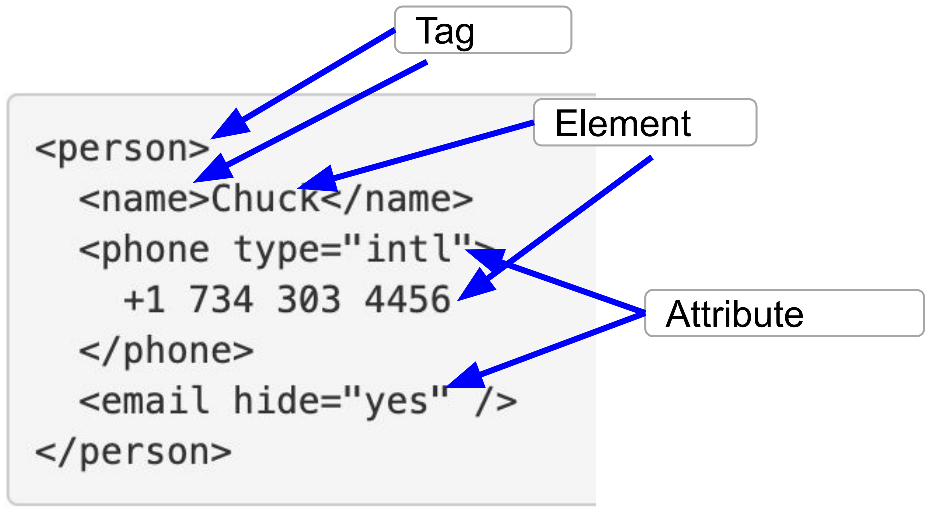
https://www.py4e.com/html3/13-web

XML

- Often it is helpful to think of an XML document as a tree structure where there is a top element (here: `person`), and other tags (e.g., `phone`) are drawn as *children* of their *parent* elements.

Representation





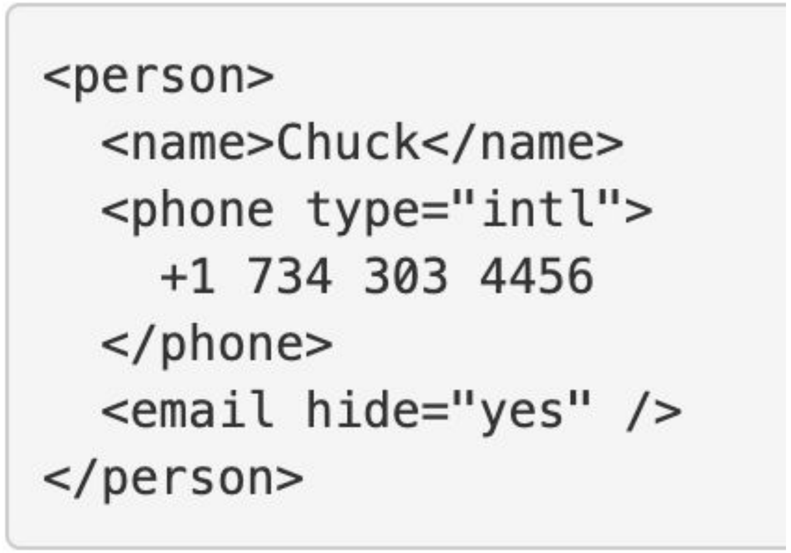
```
<person>
  <name>Chuck</name>
  <phone type="intl">
    +1 734 303 4456
  </phone>
  <email hide="yes" />
</person>
```

https://www.py4e.com/html3/13-web

XML

Tags: enclosure and label type of element

- Enclose each observation and each value for a variable.
  - Start tag e.g. <person>
  - End tag e.g. </person>
- Give variable names e.g. name, phone

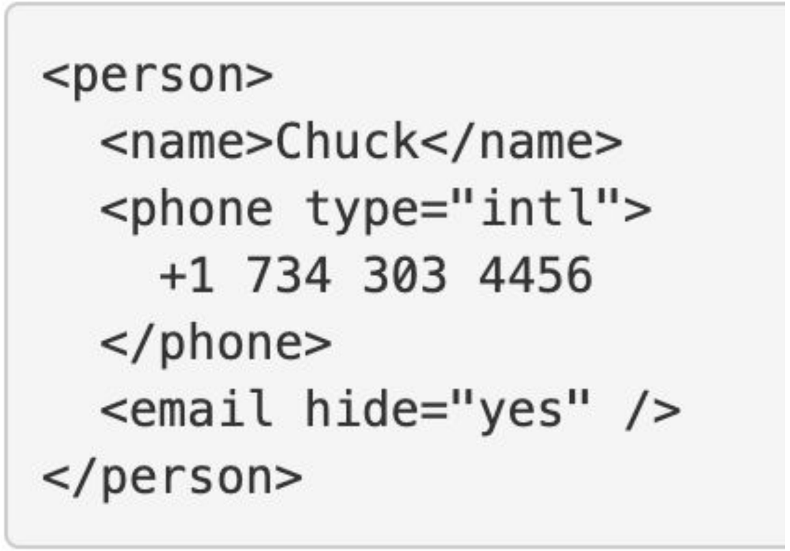


```
<person>
  <name>Chuck</name>
  <phone type="intl">
    +1 734 303 4456
  </phone>
  <email hide="yes" />
</person>
```

https://www.py4e.com/html3/13-web

XML

- Elements: tags + content between tags
  - provide data about an observation e.g. person information
  - provide data value for variable e.g. Chuck

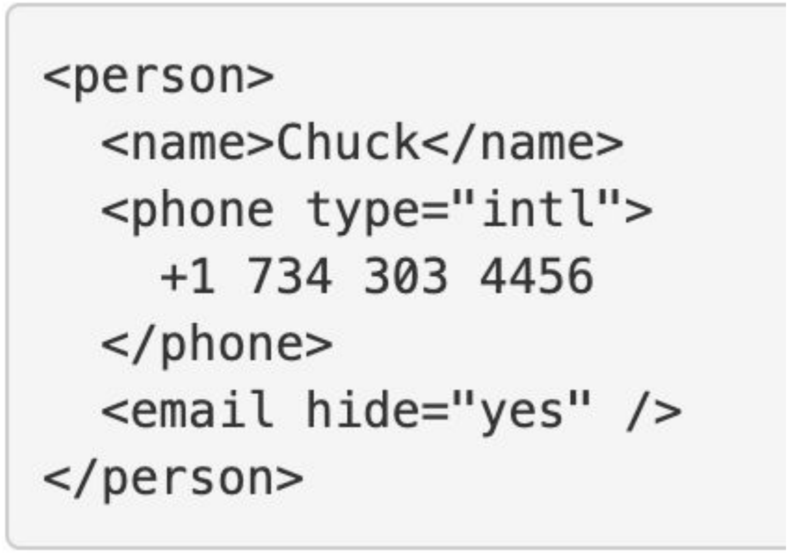


```
<person>
  <name>Chuck</name>
  <phone type="intl">
    +1 734 303 4456
  </phone>
  <email hide="yes" />
</person>
```

https://www.py4e.com/html3/13-web

XML

- Attributes: modify tag
    - define additional attributes of a variable e.g. type="intl"

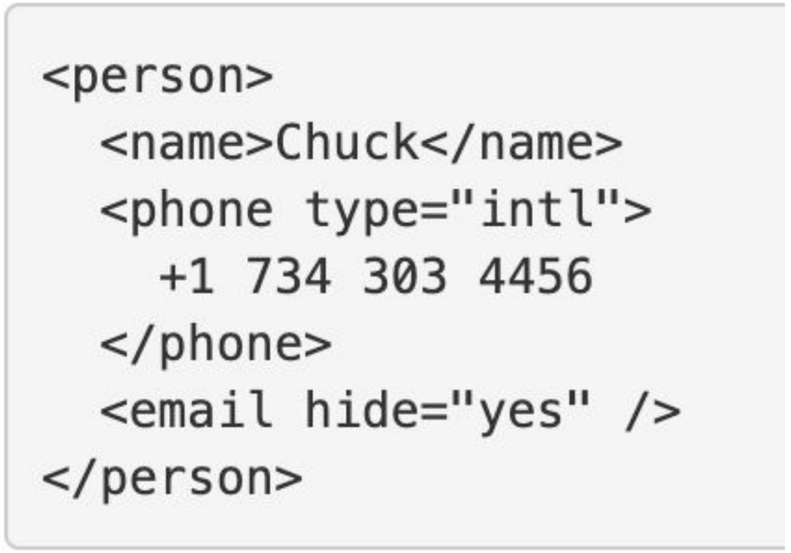


```
<person>
  <name>Chuck</name>
  <phone type="intl">
    +1 734 303 4456
  </phone>
  <email hide="yes" />
</person>
```

https://www.py4e.com/html3/13-web

XML

- Sequence of elements:
  - observations appear one after each other enclosed in tags e.g. Chuck info Roger info
  - <person>chuck..</person>
  - <person>bob..</person>

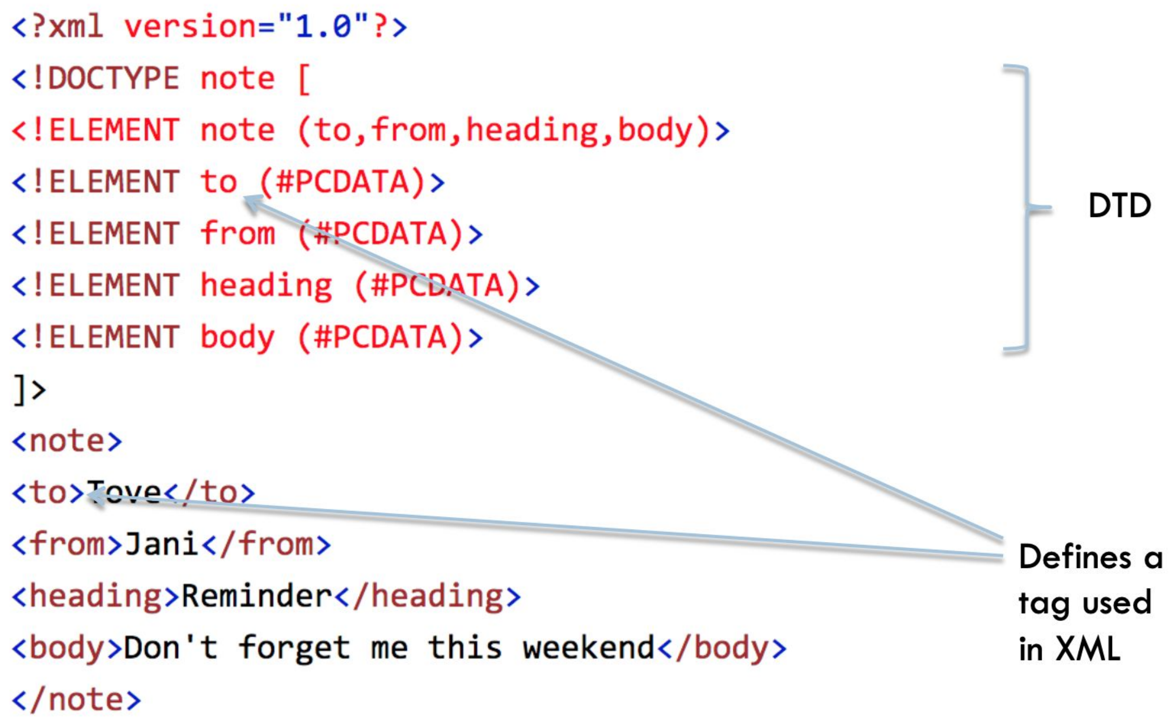


```
<person>
  <name>Chuck</name>
  <phone type="intl">
    +1 734 303 4456
  </phone>
  <email hide="yes" />
</person>
```

https://www.py4e.com/html3/13-web

XML-Document Type Definition (DTD)

- Specifies valid tags for XML

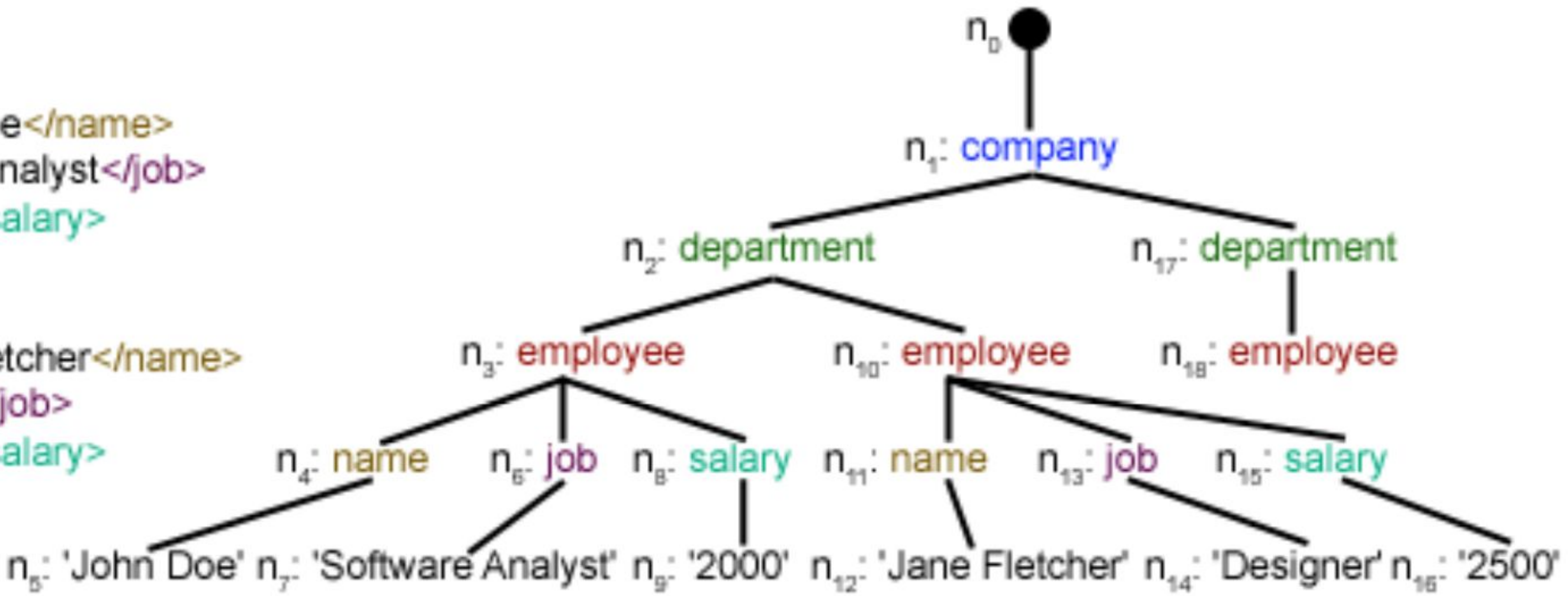


```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

https://www.py4e.com/html3/13-web

XML-Element Tree

```
<company>
 <department>
  <employee>
   <name>John Doe</name>
   <job>Software Analyst</job>
   <salary>2000</salary>
  </employee>
  <employee>
   <name>Jane Fletcher</name>
   <job>Designer</job>
   <salary>2500</salary>
  </employee>
 </department>
 <department>
  <employee>
  </employee>
 </department>
</company>
```



XML-Element Tree Functions

Read in XML as Element Tree

```
> data_etree = etree.parse(‘filename.xml’)
```

Search for specific elements by tag name

```
> listelements = data_etree.findall(“phone”)
```

XML-Element Tree Elements

Specific Element – A python object which holds information about the tag, content between tags, and attributes.

e.g. phoneelem =’<phone type =”intl”>+1 734 303 4456 </phone>’

Get tag

> phoneelem.tag

phone

Get content

> phoneelem.text

+1 734 303 4456

```
<person>
  <name>Chuck</name>
  <phone type="intl">
    +1 734 303 4456
  </phone>
  <email hide="yes" />
</person>
```

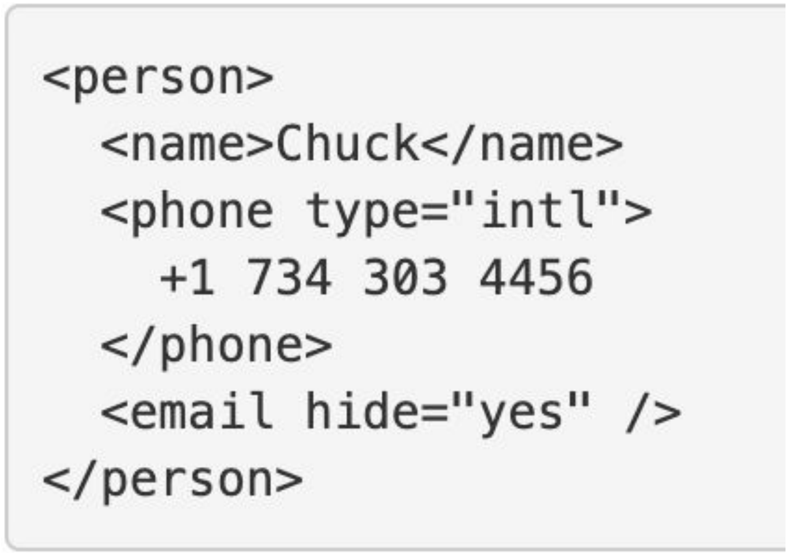
XML-Element Tree Navigation

Get parent: gets specific element that encloses the element

> personelem = phoneelem.getparent()

Get children: gets all elements enclosed by that element

> listchildren = personelem.getchildren()



```
<person>
  <name>Chuck</name>
  <phone type="intl">
    +1 734 303 4456
  </phone>
  <email hide="yes" />
</person>
```

Parsing XML

- https://www.w3schools.com/xml/xml_examples.asp
- Jupyter

JSON

JavaScript Object Notation (JSON)

- Data storage type
- Common syntax to store and transfer data across web applications
- Data is stored and transferred as text
  - lightweight
  - programming language independent

JavaScript Object Notation (JSON)

- Lists
    - Each element represents an observation e.g. person
- Dictionary
    - names represent variable names
        - e.g. phone number
    - values represent data values e.g. +1 734 303 4456

```
[{“name”:”Chuck”,
“phone”:{
  “type”: “intl”,
  “number”: “+1 734 3034456”}
“email”:{“hide”:”yes”}
},
{“name”:”Rodger”,
...
]
```

JavaScript Object Notation (JSON)

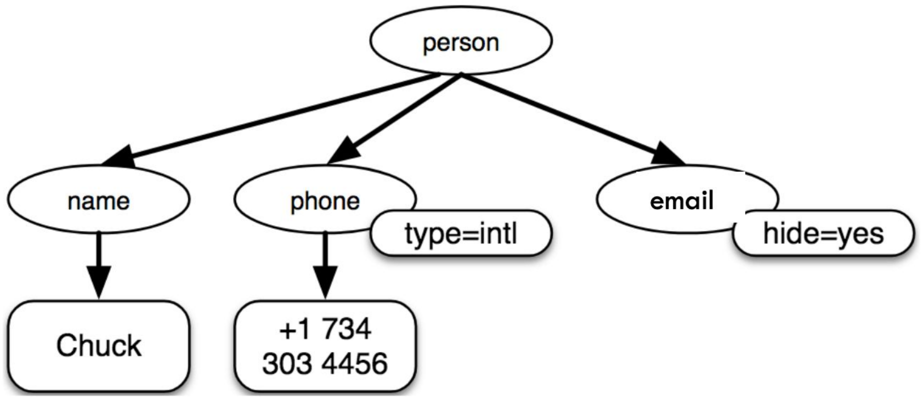
Code

```
{
  "name" : "Chuck",
  "phone" : {
    "type" : "intl",
    "number" : "+1 734 303 4456"
   },
   "email" : {
     "hide" : "yes"
   }
}
```

name

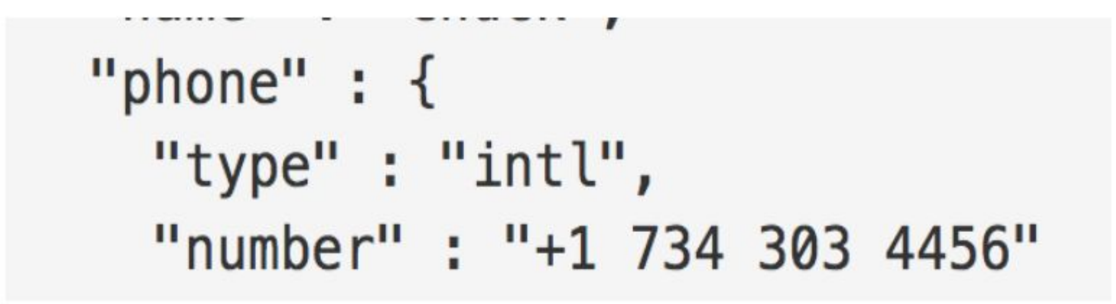
value

Representation



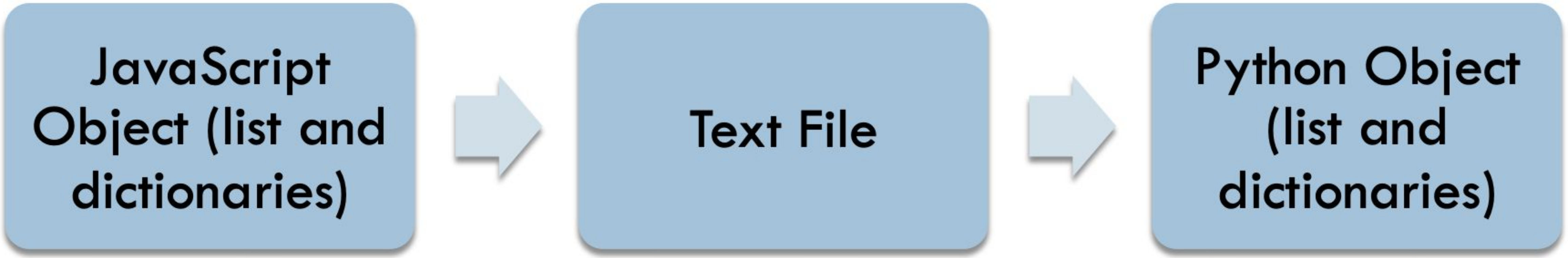
JSON - Data Storage Properties

- Hierarchical Structure:
  - Variables can be nested within other variables
    - e.g. phone has number and a type
  - Observations can be nested within other observations
    - e.g. goals are nested within match
- Inconsistent Variables
  - Variables and values may be missing for some observations
    - e.g. a match may have no goals or multiple goals



```
"phone" : {
  "type" : "intl",
  "number" : "+1 734 303 4456"
```

JSON - JSON Transfer


JavaScript Object (list and dictionaries)
Text File
Python Object (list and dictionaries)

Creating and using JSON

```
> import json
```

1. Create a string with correct syntax

```
    > ppl = ‘[{“name”:”george”},{“name”:”mary”}]’
```

2. Convert string to json to python object

```
    > pplobj = json.loads(ppl)

    > print(pplobj)

Out: [{‘name’:’george’},{‘name’:’mary’}]
```

Ingesting JSON

```
> import json
```

1. Open text file

```
> f = open(“people.json”)
```

2. Read in text file and convert to python object

```
> pplobj = json.load(f)

> print(pplobj)

Out: [{‘name’:’george’},{‘name’:’mary’}]
```

Writing JSON

> import json

1. Open text file to write

    > f = open("peoplev2.json","w")

2. Convert python object to string

    > pplstr = json.dump(pplobj)

3. Write string to text file

    > f.write(pplstr) > f.close()

JSON Properties

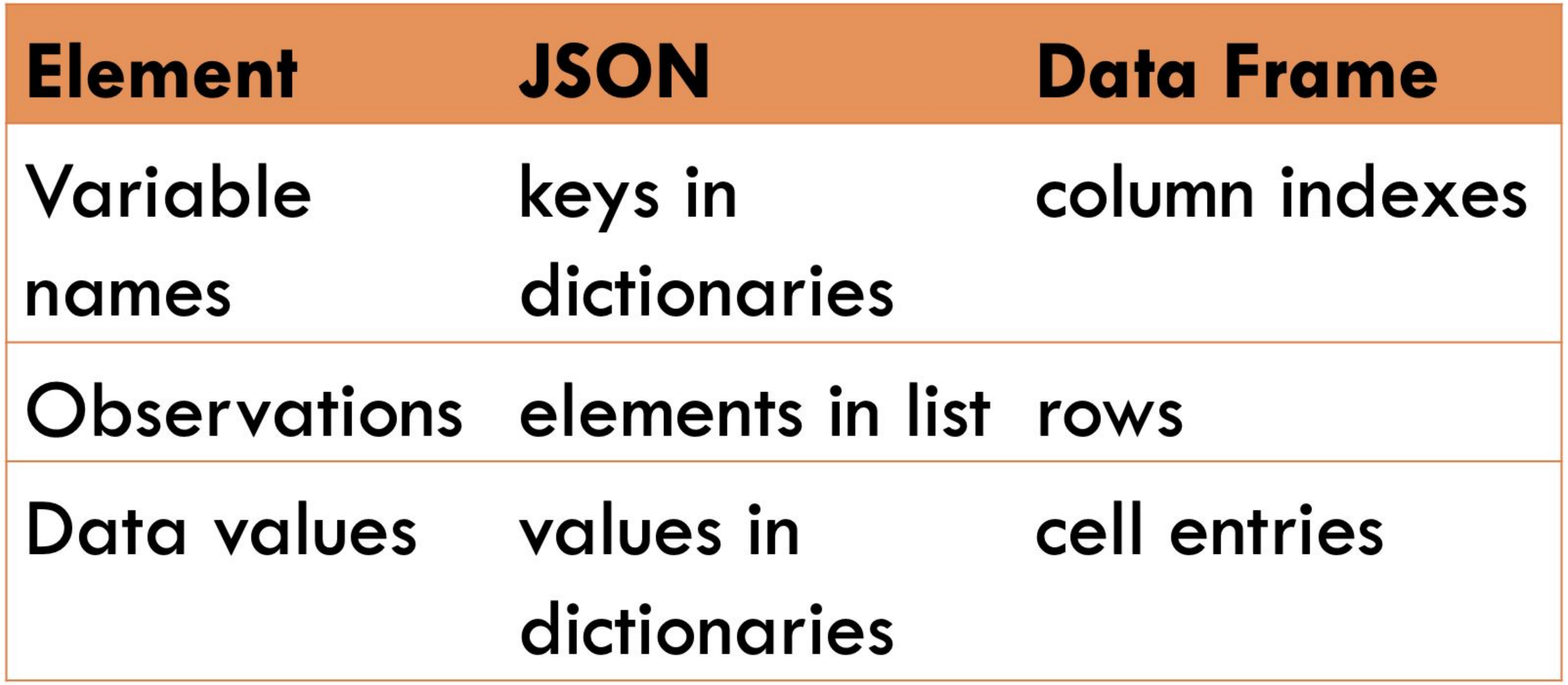
- Hierarchical Structure:
    - Variables can be nested within other variables
        - e.g. phone has number and a type
    - Inconsistent Variables
        - Variables and values may be missing for some observations
    - Not great for data analysis
        - Consistent set of non-nested variables for all observations

Normalization

- Transform JSON to DataFrame
    - Flatten – to remove hierarchical structure
    - Fill in NAs – add missing values to observations with missing entries for variables

Normalization

- Specific Transformations:



Element	JSON	Data Frame
Variable names	keys in dictionaries	column indexes
Observations	elements in list	rows
Data values	values in dictionaries	cell entries

Normalization - Example

JSON

```
[{      “name”:”John”,
        ”age”:31,
        ”city”:”NYC”},
{       “name”:”Cynthia”,
        ”age”:55,
        ”city”:”Atlanda”},
{       “name”:”Bart”,
        ”age”:10,
        ”city”:”SF”}]
```

Data Frame

name	age	city
John	31	NYC
Cynthia	55	Atlanta
Bart	10	SF

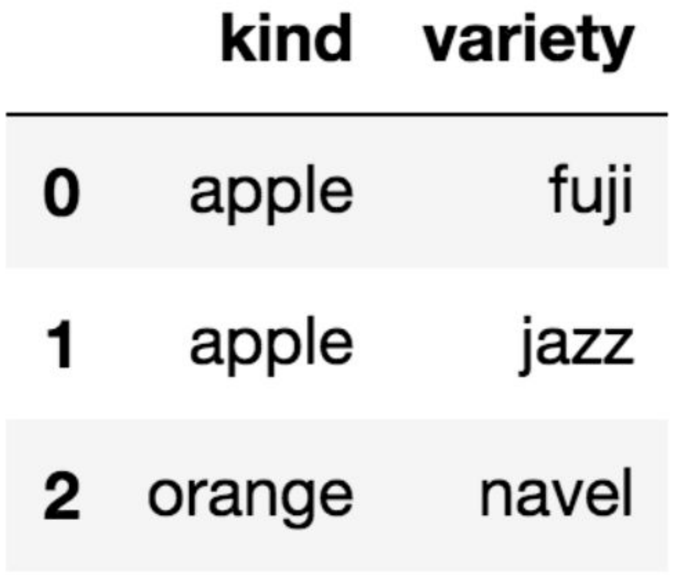
json_normalize

json_normalize(jdata, record_path, meta)

- jsondata – JSON data loaded as a python object
- record_path – one or more keys which provide a link to the list that should be converted to rows if list is within a dictionary
- meta – one or more keys which provide additional variable data, if some variables are outside of the record_path list
- from pandas.io.json import json_normalize

Example 1

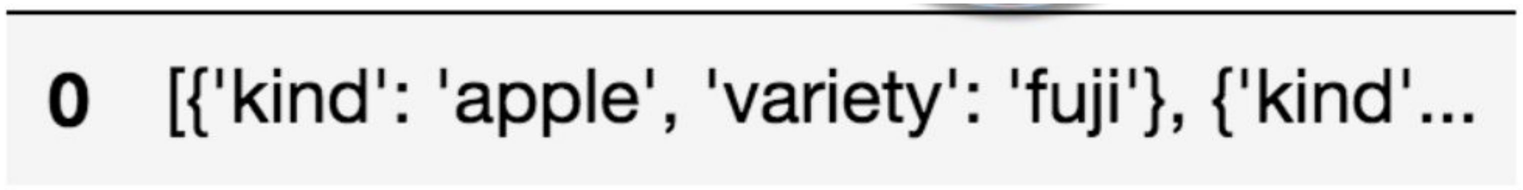
- JSON String
  - > fruits_str = '[{"kind":"apple","variety":"fuji"}, {"kind":"apple","variety":"jazz"}, {"kind":"orange","variety":"navel"}]’
- JSON Python Object
  - > fruits_json = json.loads(fruits_str)
- Data Frame
  - > fruits_df = json_normalize(fruits_json)



	kind	variety
0	apple	fuji
1	apple	jazz
2	orange	navel

Example 2

- JSON String
  - > fruits_str = ‘{“fruits”: [{"kind":"apple","variety":"fuji"}, {"kind":"apple","variety":"jazz"}, {"kind":"orange","variety":"navel"}]}’
- JSON Python Object
  - > fruits_json = json.loads(fruits_str)
- Data Frame
- > fruits_df = json_normalize(fruits_json)



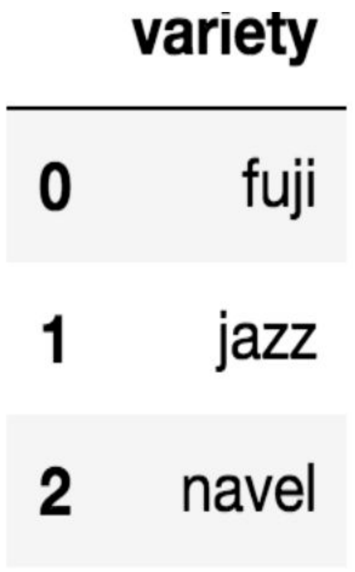
```
0   [{'kind': 'apple', 'variety': 'fuji'}, {'kind'...
```

Example 2

- JSON String
  - > fruits_str = ‘{“fruits”: [{"kind":"apple","variety":"fuji"}, {"kind":"apple","variety":"jazz"}, {"kind":"orange","variety":"navel"}]}’
- JSON Python Object
  - > fruits_json = json.loads(fruits_str)
- Data Frame
  - > fruits_df = json_normalize(fruits_json,record_path=“fruits”)

Example 3

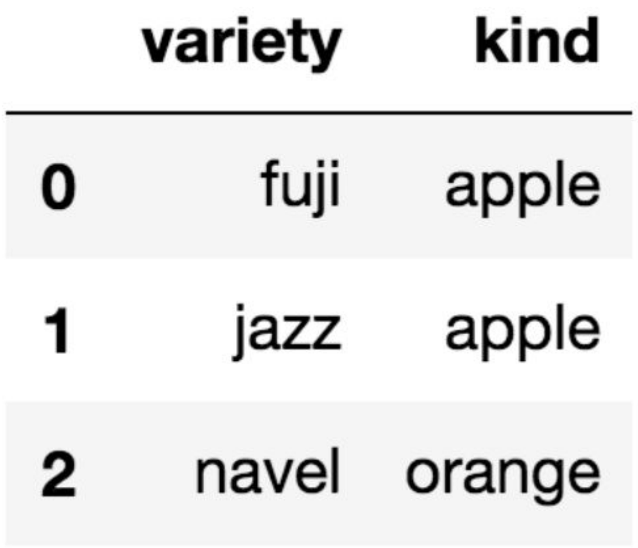
- JSON String
  - > fruits_str = '[{“kind”:”apple, “varieties”:[{"variety":"fuji"},{"variety":"jazz"}]}, {“kind”:”orange”, ”varieties”:[{"variety":"navel"}]}]’
- JSON Python Object
  - > fruits_json = json.loads(fruits_str)
- Data Frame
  - > fruits_df = json_normalize(fruits_json,record_path=“varieties”)



	variety
0	fuji
1	jazz
2	navel

Example 3

- JSON String
  - > fruits_str = '[{“kind”:”apple, “varieties”:[{"variety":"fuji"},{"variety":"jazz"}]}, {“kind”:”orange”, ”varieties”:[{"variety":"navel"}]}]’
- JSON Python Object
  - > fruits_json = json.loads(fruits_str)
- Data Frame
  - > fruits_df = json_normalize(fruits_json,record_path=“varieties”,meta=“kind”)



	variety	kind
0	fuji	apple
1	jazz	apple
2	navel	orange

Relational Databases: SQL

- Collection of one or more tables
    - Keys
        - Unique id(s)
    - Foreign keys
        - Relations between tables
    - Structured Query Language (SQL)

Similar Data Operations

Operation	Pandas	SQL
Represent data set	Data Frame	Table
Combine data sets	merge	JOIN
Aggregate data set	groupby	GROUP BY
Subset data set	iloc	WHERE

Example

- Write code to get the number of orders per aisle from the order_products data set in:
    - Pandas
    - MySQL

SQL in Python

- Connect to an existing database or create a new one
  - > con = sqlite3.connect(“database.sql”)
- Save tables to database
  - > dataframe.to_sql(“tablename”,con)
- Extract data using SQL queries
  - > newdataframe = pd.read_sql_query(query,con)

I appreciate your attention

Hope to see you on Thursday!

Reference Material
Install Software

4 Programming Assignments

- Work independently
- Deeper investigation into a data set and research question
- Turn in a well-structured and written report using Jupyter notebooks

Software Tools

- Python & Jupyter Notebook
    - Method 1
        - Python 3 (https://www.python.org/downloads)
            - Pandas Data Analysis Library (pandas)
            - Other modules (e.g. numpy, plotnine)
        - Jupyter Notebooks (aka ipython) (https://jupyter.org/install)
            - blend narrative text
            - code
            - output
            - visualizations
    - Method 2
        - Install Anaconda (includes both) (https://www.anaconda.com/distribution)
- Open Refine
    - http://openrefine.org/download.html
- Data sets
    - https://www.reddit.com/r/datasets/
    - https://opendata.dc.gov/
    - https://datasetsearch.research.google.com/
    - https://www.kaggle.com/datasets