CIS 1100

Nested Data

Python

Fall 2024

University of Pennsylvania

Dicts of Lists of Dicts of Lists of...

Not all data is *tabular*...

- sometimes we have to wrangle messy data into a `DataFrame`-y shape
  - API Responses & Data sent over the web
  - Survey responses
  - Data collected by hand
- other times we deal with data structures that aren't cleanly reducible to tables no matter what
  - family trees
  - certain file formats, like SVGs

Example: Historical Weather API

`open-meteo.com` maintains a service that lets you look up historical weather records for free using programs or a web form.

Unfortunately, when you ask it for some weather data, it looks like this:

```
{"latitude":47.90861,"longitude":-110.44777,"gener
ationtime_ms":21.31497859954834,"utc_offset_second
s":0,"timezone":"GMT","timezone_abbreviation":"GMT
","elevation":786.0,"hourly_units":{"time":"iso860
1","temperature_2m":"°F","precipitation":"inch"},"
hourly":{"time":["1972-01-15T00:00","1972-01-15T01
:00","1972-01-15T02:00","1972-01-15T03:00","1972-0
1-15T04:00","1972-01-15T05:00","1972-01-15T06:00",
"1972-01-15T07:00","1972-01-15T08:00","1972-01-15T
09:00","1972-01-15T10:00","1972-01-15T11:00","1972
-01-15T12:00","1972-01-15T13:00","1972-01-15T14:00
","1972-01-15T15:00","1972-01-15T16:00","1972-01-1
5T17:00","1972-01-15T18:00","1972-01-15T19:00","19
72-01-15T20:00","1972-01-15T21:00","1972-01-15T22:
00","1972-01-15T23:00"],"temperature_2m":[-3.3,-1.
1,2.4,7.4,11.5,14.4,17.4,20.2,23.2,25.6,29.1,31.0,
32.5,33.1,33.1,33.4,34.2,34.8,35.8,37.3,38.4,39.3,
39.5,39.6],"precipitation":[0.000,0.000,0.000,0.00
0,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,
0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.000,0.
000,0.000,0.000,0.000]}}
```

Formatting to the Rescue

```
{
    "latitude": 47.90861,
    "longitude": -110.44777,
    "generationtime_ms": 21.31497859954834,
    "utc_offset_seconds": 0,
    "timezone": "GMT",
    "timezone_abbreviation": "GMT",
    "elevation": 786,
    "hourly_units": {
        "time": "iso8601",
        "temperature_2m": "°F",
        "precipitation": "inch"
    },
    "hourly": {
        "time": ["1972-01-15T00:00", "1972-01-15T01:00", ...],
        "temperature_2m": [-3.3, -1.1, ...],
        "precipitation": [0, 0, ...]
    }
}
```

This is an example of **JSON** data

(**J**ava**s**cript **O**bject **N**otation)

JSON

JSON is a common standard for data returned from requests made on the internet.

- Not just for Javascript
- In fact, it looks a lot like a Python dictionary…

```json
{
    "latitude": 47.90861,
    "longitude": -110.44777,
    "generationtime_ms": 21.31497859954834,
    "utc_offset_seconds": 0,
    "timezone": "GMT",
    "timezone_abbreviation": "GMT",
    "elevation": 786,
    "hourly_units": {
        "time": "iso8601",
        "temperature_2m": "°F",
        "precipitation": "inch"
    },
    "hourly": {
        "time": ["1972-01-15T00:00", "1972-01-15T01:00", ...],
        "temperature_2m": [-3.3, -1.1, ...],
        "precipitation": [0, 0, ...]
    }
}
```

CIS 1100

JSON & Python

Python
Fall 2024
University of Pennsylvania

`json`

Python has a built-in JSON encoding and decoding library called `json`.

`json.loads()` (*"load string"*) parses a string of JSON data into a dictionary.

```
import json
data_string = '{"name" : "Harry"}'
data_dict = json.loads(data_string)
print(data_dict)
print(data_dict["name"])
```

🖨️👇

```
{'name': 'Harry'}
Harry
```

json

`json.load()` parses a string of JSON data into a dictionary.

```
import json
data_file = open('weather_response.json', 'r')
data_dict = json.load(data_file)
print(data_dict.keys())
```

🖨️👇

```
dict_keys(['latitude', 'longitude',
'generationtime_ms', 'utc_offset_seconds',
'timezone', 'timezone_abbreviation',
'elevation', 'hourly_units', 'hourly'])
```

Traversing Through JSON Data

Here are our keys…

```
dict_keys(['latitude', 'longitude',  'generationtime_ms', 'utc_offset_seconds',
'timezone', 'timezone_abbreviation', 'elevation', 'hourly_units', 'hourly'])
```

But how do we read it? And where are time and temperature in the dictionary?

```
{
    "latitude": 47.90861,
    "longitude": -110.44777,
    "generationtime_ms": 21.31497859954834,
    "utc_offset_seconds": 0,
    "timezone": "GMT",
    "timezone_abbreviation": "GMT",
    "elevation": 786,
    "hourly_units": {
        "time": "iso8601",
        "temperature_2m": "°F",
        "precipitation": "inch"
    },
    "hourly": {
        "time": ["1972-01-15T00:00", "1972-01-15T01:00", ...],
        "temperature_2m": [-3.3, -1.1, ...],
        "precipitation": [0, 0, ...]
    }
}
```

Nesting in JSON Data

A JSON Object will often have a nested, hierarchical structure.

- Some keys in the dictionary map to primitives

```
"latitude": 47.90861,
"longitude": -110.44777,
"generationtime_ms": 21.31497859954834,
"utc_offset_seconds": 0,
```

- Other keys map to other dictionaries...

```
"hourly_units": {
    "time": "iso8601",
    "temperature_2m": "°F",
    "precipitation": "inch"
},
```

Nesting in JSON Data

A JSON Object will often have a nested, hierarchical structure.

- And sometimes those dictionaries store other dictionaries or lists themselves!

```
"hourly": {
    "time": ["1972-01-15T00:00", "1972-01-15T01:00", ...],
    "temperature_2m": [-3.3, -1.1, ...],
    "precipitation": [0, 0, ...]
}
```

Working with Nested Structures

Answering questions using nested structures/JSON often requires...

- careful study of the structure by looking at keys, brackets
- list indexing and dictionary lookups, which you already know how to do!

Answering Questions

Where is this weather sample taken from?

```
{
    "latitude": 47.90861,
    "longitude": -110.44777,
    "generationtime_ms": 21.31497859954834,
    "utc_offset_seconds": 0,
    "timezone": "GMT",
    "timezone_abbreviation": "GMT",
    "elevation": 786,
    "hourly_units": {
        "time": "iso8601",
        "temperature_2m": "°F",
        "precipitation": "inch"
    },
    "hourly": {
        "time": ["1972-01-15T00:00", "1972-01-15T01:00", ...],
        "temperature_2m": [-3.3, -1.1, ...],
        "precipitation": [0, 0, ...]
    }
}
```

Answering Questions

Where is this weather sample taken from?

```python
data = json.load(weather_file)
lat = data["latitude"]
lon = data["longitude"]
elv = data["elevation"]
print(f"Sample is from coordinates ({lat}, {lon}) at elevation of {elv} feet.")
```

🖨️👇

```
Sample is from coordinates (39.964848, 75.2933) at elevation of 2924.0 feet.
```

Answering Questions

Are the weather samples in Farenheit or Celsius?

```
{
    "latitude": 47.90861,
    "longitude": -110.44777,
    "generationtime_ms": 21.31497859954834,
    "utc_offset_seconds": 0,
    "timezone": "GMT",
    "timezone_abbreviation": "GMT",
    "elevation": 786,
    "hourly_units": {
        "time": "iso8601",
        "temperature_2m": "°F",
        "precipitation": "inch"
    },
    "hourly": {
        "time": ["1972-01-15T00:00", "1972-01-15T01:00", ...],
        "temperature_2m": [-3.3, -1.1, ...],
        "precipitation": [0, 0, ...]
    }
}
```

Answering Questions

Are the weather samples in Farenheit or Celsius?

```
data = json.load(weather_file)
units = data["temperature_2m"]
```

✗✗✗✗✗✗✗✗✗✗✗✗✗✗

Answering Questions

Are the weather samples in Farenheit or Celsius?

```python
data = json.load(weather_file)
units = data["hourly_units"]
degrees = units["temperature_2m"]
print(f"Temperature given in {degrees}.")
```

🖨️👇

```
Temperature given in °F.
```

Answering Questions

Are the weather samples in Farenheit or Celsius?

```
data = json.load(weather_file)
degrees = data["hourly_units"]["temperature_2m"]
print(f"Temperature given in {degrees}.")
```

🖨️ 👇

```
Temperature given in °F.
```

Answering Questions

How many temperature samples are included?

What was the range of temperatures measured?

```json
{
    "latitude": 47.90861,
    "longitude": -110.44777,
    "generationtime_ms": 21.31497859954834,
    "utc_offset_seconds": 0,
    "timezone": "GMT",
    "timezone_abbreviation": "GMT",
    "elevation": 786,
    "hourly_units": {
        "time": "iso8601",
        "temperature_2m": "°F",
        "precipitation": "inch"
    },
    "hourly": {
        "time": ["1972-01-15T00:00", "1972-01-15T01:00", ...],
        "temperature_2m": [-3.3, -1.1, ...],
        "precipitation": [0, 0, ...]
    }
}
```

Answering Questions

How many temperature samples are included?

What was the range of temperatures measured?

```python
data = json.load(weather_file)
degrees = data["hourly_units"]["temperature_2m"]
temperatures = data["hourly"]["temperature_2m"]
num_samples = len(temperatures)
low, high = min(temperatures), max(temperatures)
temp_range = high - low
print(f"Over {num_samples} samples,")
print(f"the temperature shifted from {high} to {low}.")
print(f"That's a swing of {temp_range} {degrees}!")
```

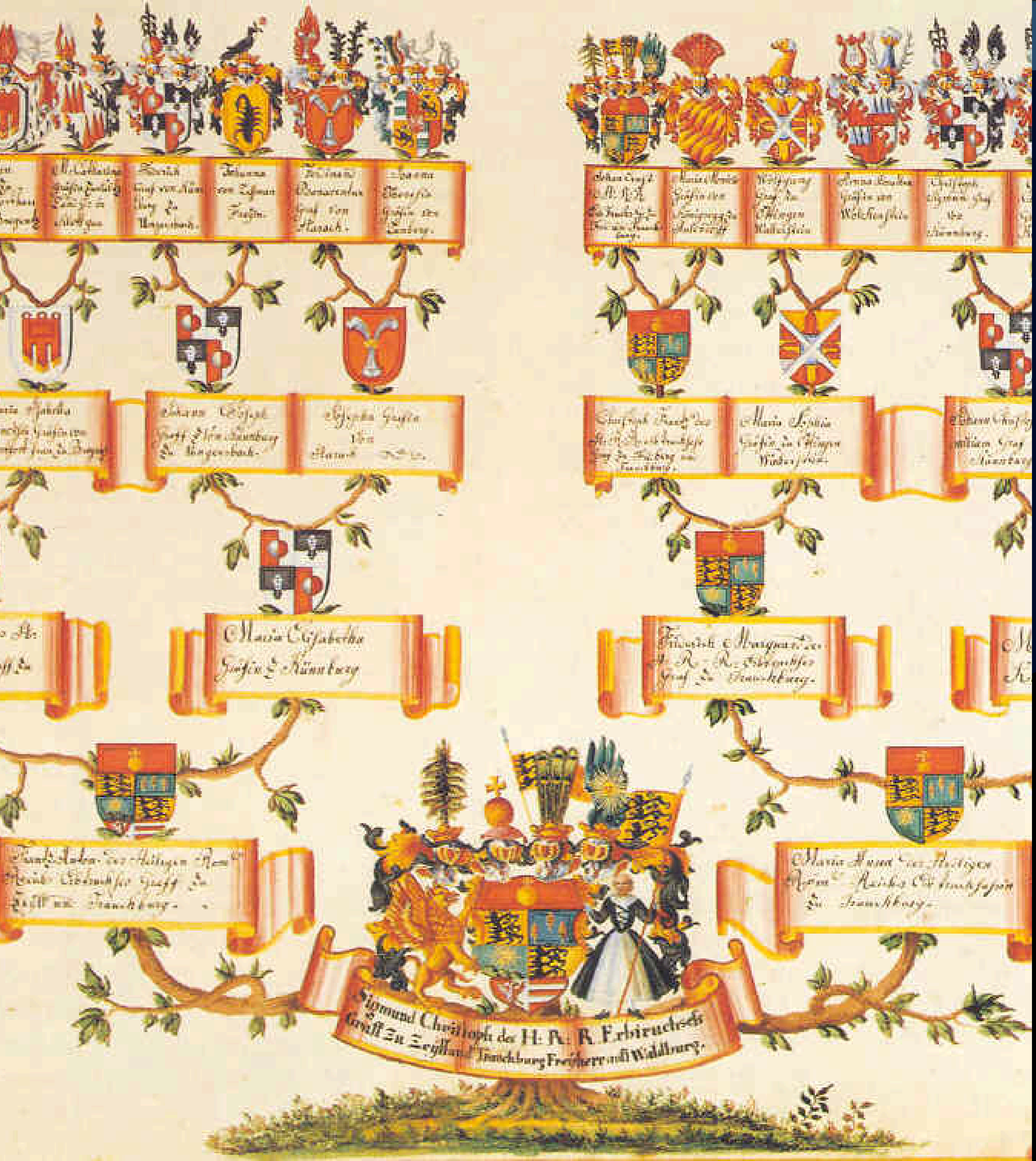
🖨️👇

```
Over 360 samples,
the temperature shifted from 59.0 to 31.2.
That's a swing of 27.8 °F!
```

CIS 1100

Trees & XML

Python
Fall 2024
University of Pennsylvania


Sigmund Christoph des H. R. R. Erbtruchseß
Graff zu Zeyll und Trauchburg Freyherr auf Waldburg

Tree Data

Trees in computer science are hierarchical collections of data elements (often called **nodes**) that have connections between them.

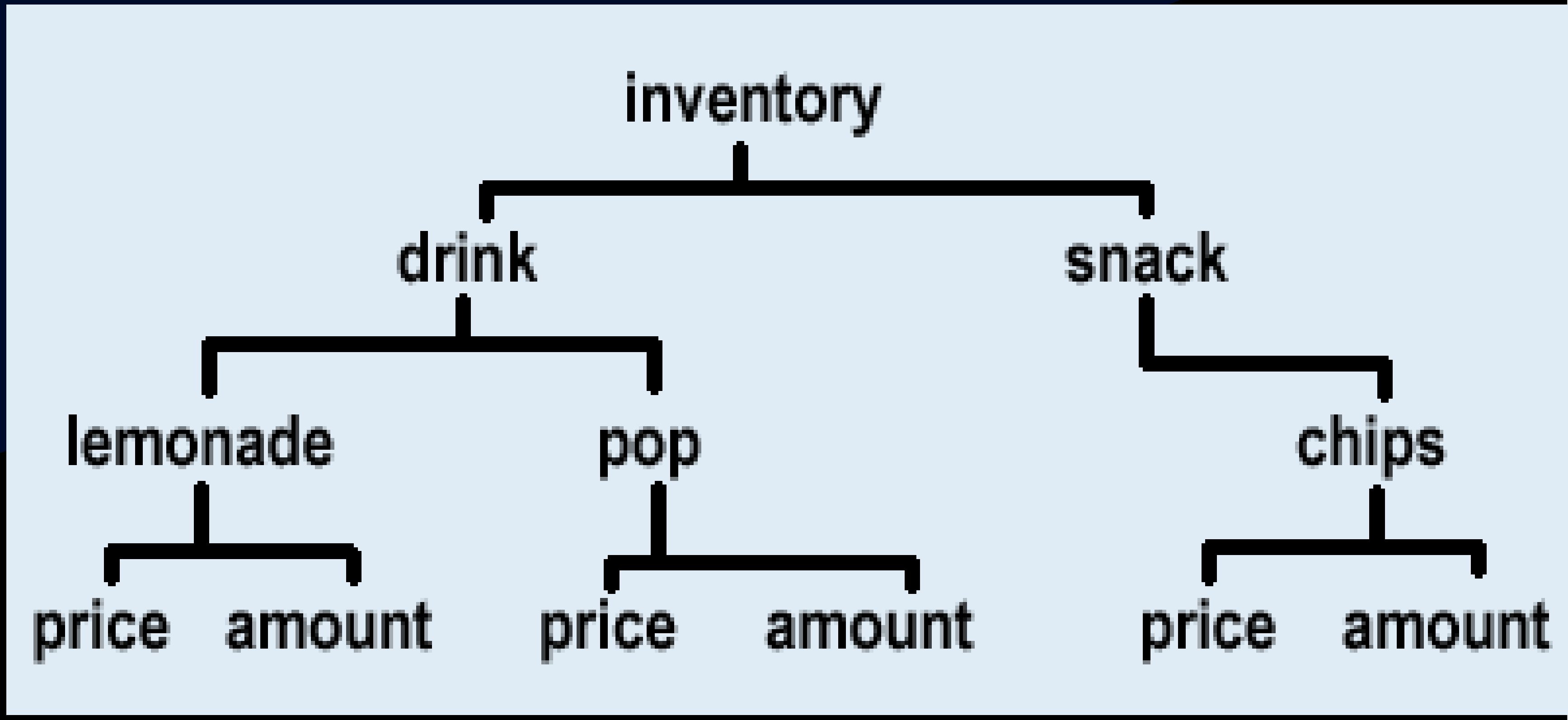
- e.g. family trees

XML

XML is a data document format that allows us to represent data nested inside of other data.

```
<inventory>
        <drink>
                <lemonade>
                        <price>$2.50</price>
                        <amount>20</amount>
                </lemonade>
                <pop brand="Pepsi">
                        <price>$1.50</price>
                        <amount>10</amount>
                </pop>
        </drink>

        <snack>
                <chips flavor="BBQ">
                        <price>$4.50</price>
                        <amount>60</amount>
                </chips>
        </snack>
</inventory>
```



http://www.tizag.com/xmlTutorial/xmltree.php

Some XML Terminology

- **Elements** are the entities being represented in the XML tree, e.g. an inventory or a price.
- **Tags** are the names that we give to the elements, e.g. `<inventory>` or `<price>`
- **Attributes** are properties that individual elements can have, stored in the tags
    - ○ If the pop element is specifically a Pepsi, we could have its tag be `<pop brand="Pepsi">`.

```
<inventory>
        <drink>
                <lemonade>
                        <price>$2.50</price>
                        <amount>20</amount>
                </lemonade>
                <pop brand="Pepsi">
                        <price>$1.50</price>
                        <amount>10</amount>
                </pop>
        </drink>

        <snack>
                <chips flavor="BBQ">
                        <price>$4.50</price>
                        <amount>60</amount>
                </chips>
        </snack>
</inventory>
```

Some Tree Terminology

- The **tree** is the collection of elements being represented and the connections between them
- The **root** is the element of the tree that has no ancestors.
- An **ancestor** is an element that contains another element.
    - A **parent** is a direct ancestor.
- A **descendant** is an element that is contained by another element.
    - A **child** is a direct descendant.

```
<inventory>
        <drink>
                <lemonade>
                        <price>$2.50</price>
                        <amount>20</amount>
                </lemonade>
                <pop brand="Pepsi">
                        <price>$1.50</price>
                        <amount>10</amount>
                </pop>
        </drink>

        <snack>
                <chips flavor="BBQ">
                        <price>$4.50</price>
                        <amount>60</amount>
                </chips>
        </snack>
</inventory>
```

Why XML?

It's a convenient standard for representing hierarchy.

- Family trees, inventory systems
- Degree requirements & course dependencies

Many visual elements are best represented as hierarchical data

- PowerPoints and Word Documents are actually just XML documents rendered in a fancy way
    - Slides, which have boxes, which have images & text, which each have properties...
- Some image formats are XML, like SVG

```
<ns0:svg xmlns:ns0="http://www.w3.org/2000/svg" id="emoji" viewBox="0 0 72 72">
  <ns0:g id="color">
    <ns0:path fill="#d0cfce" d="m56..." />
    <ns0:path fill="#9b9b9a" d="m36.."/>
  </ns0:g>
  <ns0:g id="line">
    <ns0:path fill="none" stroke="#000" stroke-linecap="round" ... />
    <ns0:path fill="none" stroke="#000" stroke-miterlimit="10" ... />
  </ns0:g>
</ns0:svg>
```

👽 👽 👽 👽

CIS 1100

XML & Python

Python
Fall 2024
University of Pennsylvania

Parsing XML with Python

`xml.etree.ElementTree` is the built-in class we use to parse or construct XML:

```python
import xml.etree.ElementTree as ET
tree = ET.parse('country_data.xml') # parse is used for reading a file
root = tree.getroot()               # usually we work with the root directly
```

Demo XML

This is `country_data.xml`:

```xml
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
```

https://docs.python.org/3/library/xml.etree.elementtree.html

Parsing XML with Python

The `root` is an `Element`, which stores its tag name and a dictionary of its attributes:

```
>>> root.tag
'data'
>>> root.attrib
{}
```

"The root is a 'data' element that stores no attributes."

Look at the Children

You can iterate over an `Element`, which visits each of its children.

```python
for child in root:
    print(child.tag, child.attrib)
```

🖨️👇

```
country {'name': 'Liechtenstein'}
country {'name': 'Singapore'}
country {'name': 'Panama'}
```

```xml
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
```

Looking Further

You can search over all descendants of an element that have a specific tag using `.iter(tag_name)`

```python
for neighbor in root.iter('neighbor'):
    print(neighbor.attrib)
```

🖨️👇

```
{'name': 'Austria', 'direction': 'E'}
{'name': 'Switzerland', 'direction': 'W'}
{'name': 'Malaysia', 'direction': 'N'}
{'name': 'Costa Rica', 'direction': 'W'}
{'name': 'Colombia', 'direction': 'E'}
```

```xml
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
```

Filtering Children

`.findall(tag_name)` gives all children of a given `Element` that have a matching tag.
`.find(tag_name)` gives the first child of a given `Element` that have a matching tag.

```python
for country in root.findall('country'):
    rank = country.find('neighbor')
    print(rank)
```

🖨️ 👇

```
{'name': 'Austria', 'direction': 'E'}
{'name': 'Malaysia', 'direction': 'N'}
{'name': 'Costa Rica', 'direction': 'W'}
```

```xml
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
```