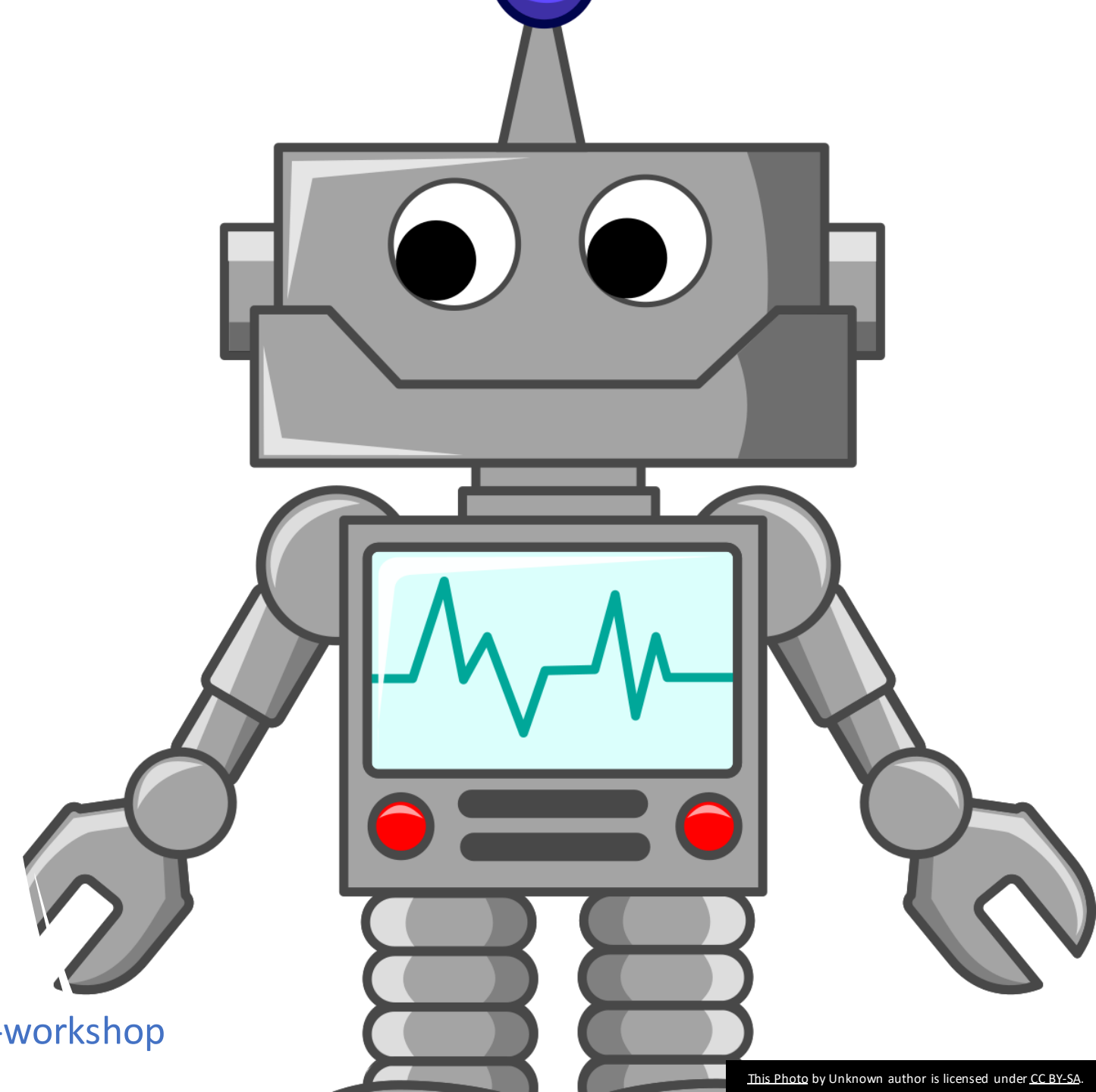
Python Workshop I: Machine Readable Formats

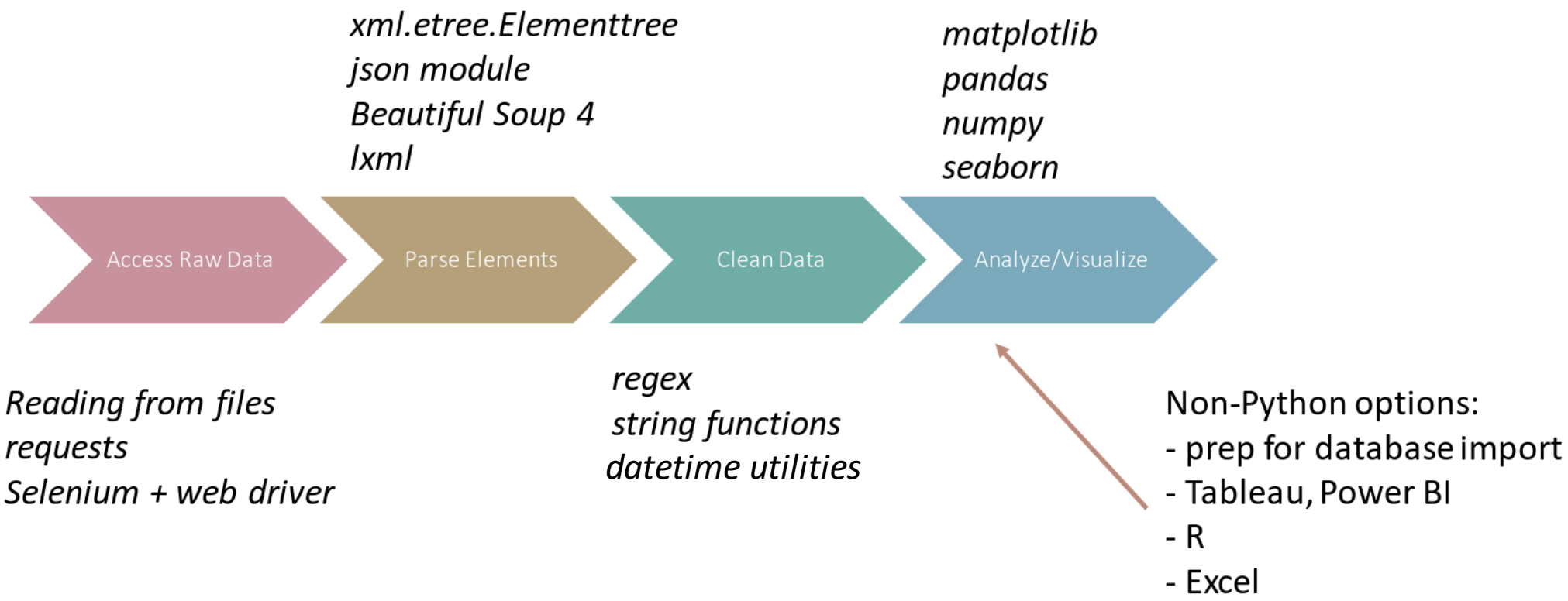
Monica Ihli

monica@utk.edu

https://github.com/monicaihli/python-data-workshop



Data Processing Workflow



1. Get the raw data on your computer.
2. Extract into Python data structures
3. Use Python tools to clean up and restructure usable format.
4. Analyze the data

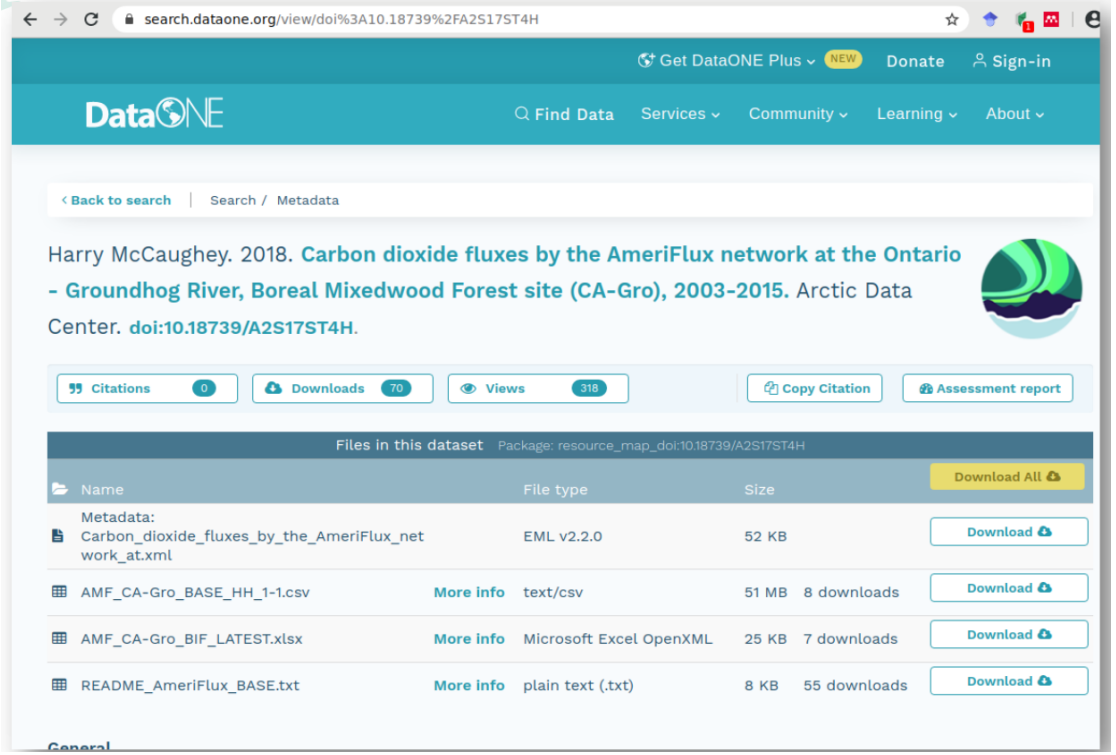
Who is This Data For?

Human-Readable Content

- Content is often navigated to (like websites).
- Presentation of content prioritizes visual appeal.
- Information is embedded in mark-up code such as HTML.
- Documents may be structured, but content is not.
- Often more work to extract and transform to orderly and native Python data types and structures.

Machine-Readable Content

- Often exposed through dedicated web service applications and locations.
- Prioritizes consistency and structure. Encoding is structural.
- Can be ingested / layered with other applications or encoding to transform raw data into aesthetically pleasing presentation. (e.g., XLST, opening data in Excel)
- Examples: text-based formats (csv, tsv), proprietary spreadsheets, XML, JSON, etc.




```
-<eml:eml packageId="doi:10.18739/A2S17ST4H" xsi:schemaLocation="https://eml.ecoinformatics.org/eml-2.2.0 eml.xsd" scope="system" sy
 -<dataset>
   <alternateIdentifier system="DOI">10.17190/AMF/1245996</alternateIdentifier>
  -<title>
     Carbon dioxide fluxes by the AmeriFlux network at the Ontario - Groundhog River, Boreal Mixedwood Forest site (CA-Gro), 2003-2015
   </title>
  -<creator>
   -<individualName>
      <givenName>Harry</givenName>
      <surName>McCaughey</surName>
    </individualName>
    <organizationName>Queen's University</organizationName>
    <electronicMailAddress>mccaughe@queensu.ca</electronicMailAddress>
    <userId directory="https://orcid.org">https://orcid.org/0000-0003-0896-1255</userId>
  </creator>
  <pubDate>2018</pubDate>
  -<abstract>
     Dynamics of carbon dioxide, water, and energy fluxes and the associated meteorological drivers; assessment of the ecological character of t
     sensing to model carbon dioxide exchange and the status of macronutrients in the canopy.
  </abstract>
  -<intellectualRights>
   -<para>
      This work is licensed under the Creative Commons Attribution 4.0 International License.To view a copy of this license, visit http://creative
    </para>
  </intellectualRights>
  -<distribution>
   -<online>
      <url>http://doi.org/doi:10.18739/A2S17ST4H</url>
    </online>
  </distribution>
  -<coverage>
   -<geographicCoverage>
    -<geographicDescription>
       Groundhog River (FCRN or CCP site 'ON-OMW') is situated in a typical boreal mixedwood forest in northeastern Ontario (48.217 degre
```

Tabular Data

csv

tsv

spreadsheets

txt

Tabular Data

- Data is stored in rows and columns.
- Typically, each observation/case is a row, and each column is a variable that takes on a value.
- "Delimited" files include .CSV (comma-separated value), .TSV (tab-separated value), etc.
- Spreadsheet applications such as Excel or LibreCalc or Open Office, produce proprietary versions such as .xlsx.

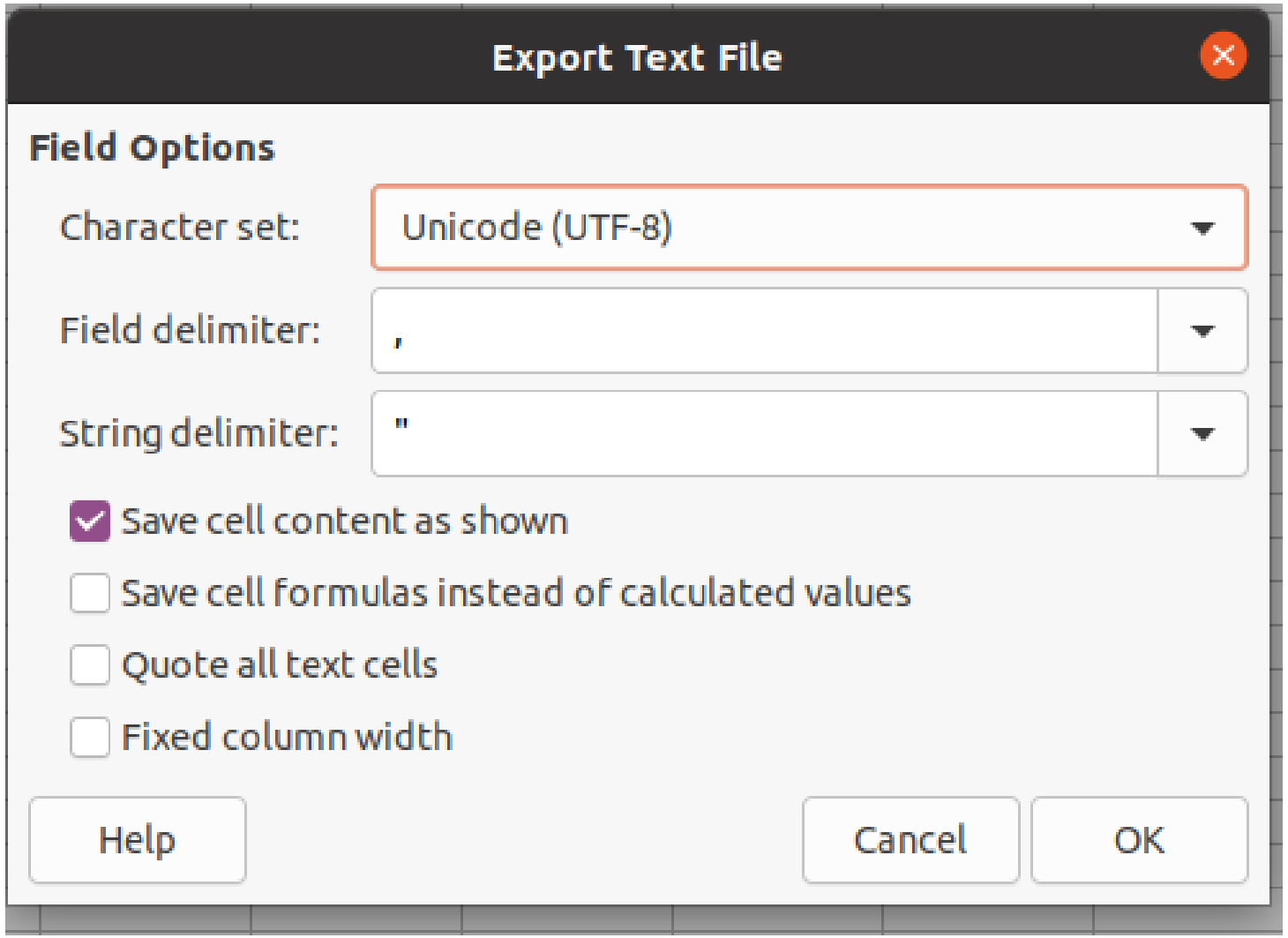
	A	B	C
1	Dept	Course	Title
2	ANTH	600	Big Data for Social Sciences
3	ECE	526	Data mining
4	INSC	584	Database Management Systems
5	STAT	537	Statistics for Research I
6			

data_courses.csv

Creating & Consuming Data Files

- The decisions made when creating data affect how data must be parsed.

- Possible Considerations:
  - Character encoding
  - Field delimiters
  - Line terminators
  - Quoted cell values
  - Escape Characters



Saving data in a spreadsheet to a CSV file.

CSV Module
(Python Standard Library)

csv.reader()

map each line to a list of strings

```
import csv
with open('data_courses.csv', encoding='utf-8') as csvfile:
    my_csv_reader = csv.reader(csvfile)
```

```
my_csv_reader = {reader} <_csv.reader object at 0x7fd550a4d890>
    dialect = {Dialect} <_csv.Dialect object at 0x7fd550b5f870>
        delimiter = {str} ','
        doublequote = {bool} True
        escapechar = {NoneType} None
        lineterminator = {str} '\r\n'
        quotechar = {str} '"'
        quoting = {int} 0
        skipinitialspace = {bool} False
        strict = {bool} False
    line_num = {int} 0
```

- Many of these properties show how the reader object has been configured to read the file. These are **parameters** you can adjust for reading the file.
- The **line_num** value reports how far you've proceeded into the file so far. The line number will increment each time you read a line.

Iterate over each line

```
import csv
with open('data_courses.csv', encoding='utf-8') as csvfile:
    my_csv_reader = csv.reader(csvfile)
    for row in my_csv_reader:
        print(row)
```

```
row = {list: 4} ['Dept', 'Course', 'Title', 'Credits']
    0 = {str} 'Dept'
    1 = {str} 'Course'
    2 = {str} 'Title'
    3 = {str} 'Credits'
    __len__ = {int} 4
```

```
1
['Dept', 'Course', 'Title', 'Credits']
2
['ANTH', '600', 'Big Data for Social Sciences', '3']
3
['ECE', '526', 'Data mining', '3']
4
['INSC', '584', 'Database Management Systems', '3']
5
['STAT', '537', 'Statistics for Research I', '3']
```

csv_parser.py, Example # 1

Code

```
with open('data_courses.csv', encoding='utf-8') as csvfile:
    my_csv_reader = csv.reader(csvfile)
    next(my_csv_reader) # skip the header row
    total_credits = 0

    for row in my_csv_reader:
        print(row[0] + ": " + row[1] + " - " + row[2])
        total_credits += int(row[3])

print("These courses add up to " + str(total_credits) + "
credit hours.")
```

Output:

```
ANTH: 600 - Big Data for Social Sciences
ECE: 526 - Data mining
INSC: 584 - Database Management Systems
STAT: 537 - Statistics for Research I
These courses add up to 12 credit hours.


Process finished with exit code 0
```

Loading the data into Python data structures & types means we can use all the tools and utilities within Python to manipulate, transform, and restructure that data.

csv.DictReader()

map each line to a dict instead of a list

```
import csv
with open('data_courses.csv', encoding='utf-8') as csvfile:
    my_csv_reader = csv.reader(csvfile)
    for row in my_csv_reader:
        print(row)
```

```
row = {dict: 4} {'Dept': 'ANTH', 'Course': '600', 'Title': 'Big Data for Social Sciences', 'Credits': '3'}
    'Dept' = {str} 'ANTH'
    'Course' = {str} '600'
    'Title' = {str} 'Big Data for Social Sciences'
    'Credits' = {str} '3'
    __len__ = {int} 4
```

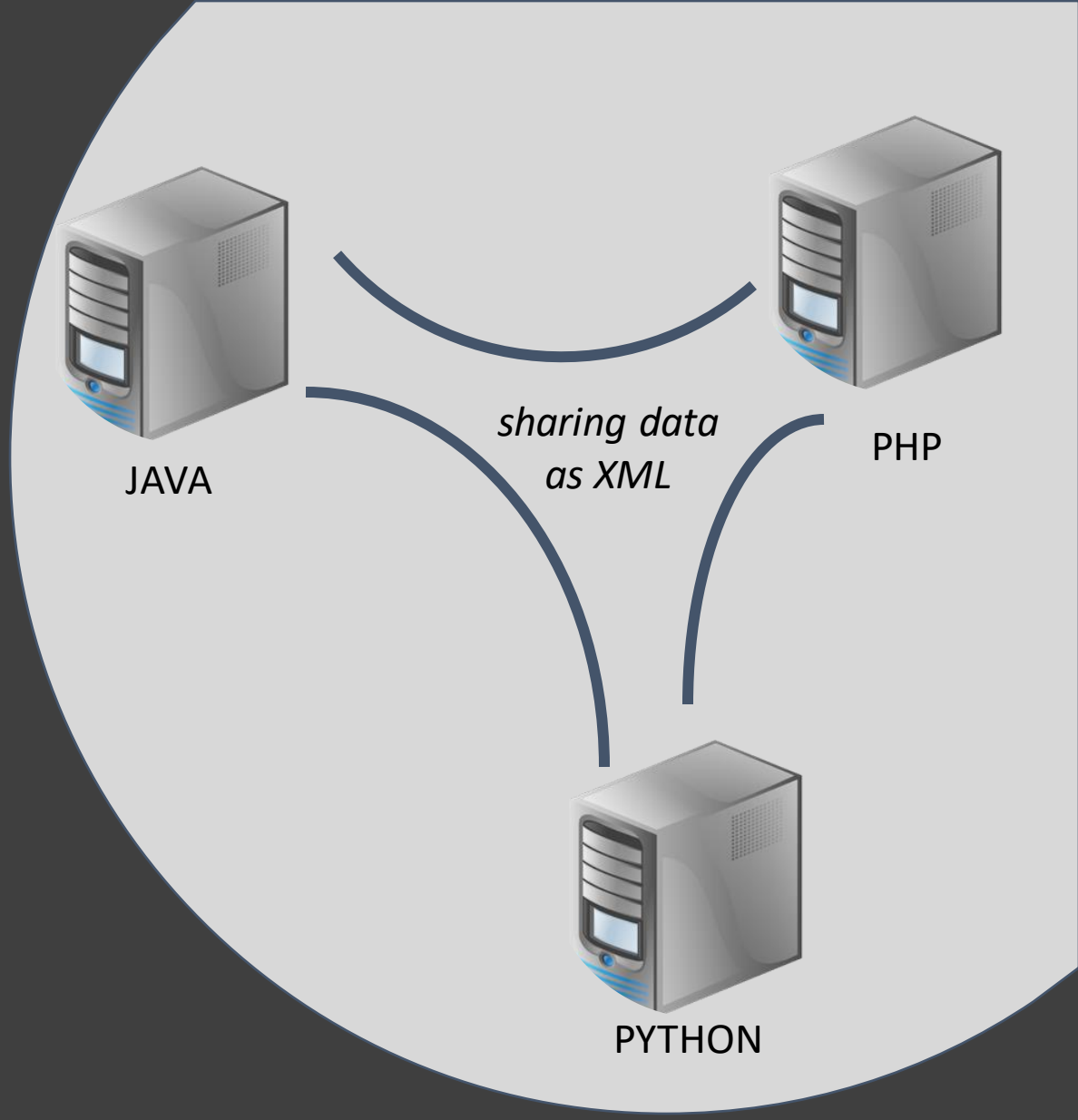
```
1
['Dept', 'Course', 'Title', 'Credits']
2
['ANTH', '600', 'Big Data for Social Sciences', '3']
3
['ECE', '526', 'Data mining', '3']
4
['INSC', '584', 'Database Management Systems', '3']
5
['STAT', '537', 'Statistics for Research I', '3']
```

csv_to_dict_parser.py, Example # 1

Getting Started with XML

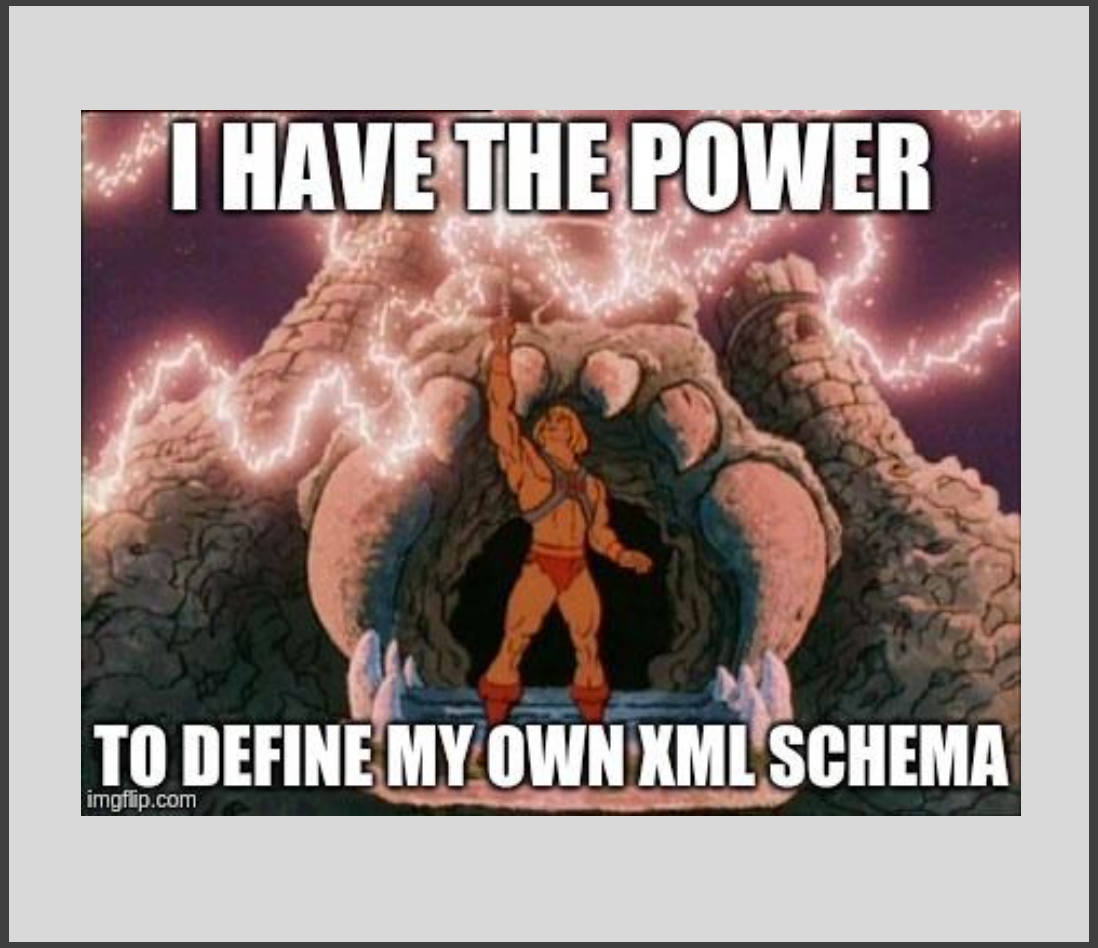
Interoperability

- Language agnostic syntax.
- Different languages have their own parsers and other libraries for working with XML.



Structured but Flexible

- Fully customizable approach to organizing data.
- Proprietary formats can be enforced via schemas and schema validation.



XML Elements

- Elements are the primary component of an XML document.
- They store information.
- You can call them anything you want!
- Syntax:

```
<my_element>My contents</my_element>
```

- Rules for naming elements:
  - Names are case-sensitive.
  - May contain letters, hyphens, hyphens, underscores, and periods.
  - Must start with a letter or underscore.
  - May not contain spaces.
  - May not start with the letters xml.

Naming Elements

elements

- <xs:element name="cat">
- Elements are the building blocks of an XML document.

ATTRIBUTES

```
<cat id="1">
```

- XML elements can have attributes.
- Attributes are embedded in line with the XML.
- Attribute values are enclosed in quotes.
- Attributes vs. additional elements in a complex type are largely a matter of preference.

ATTRIBUTES

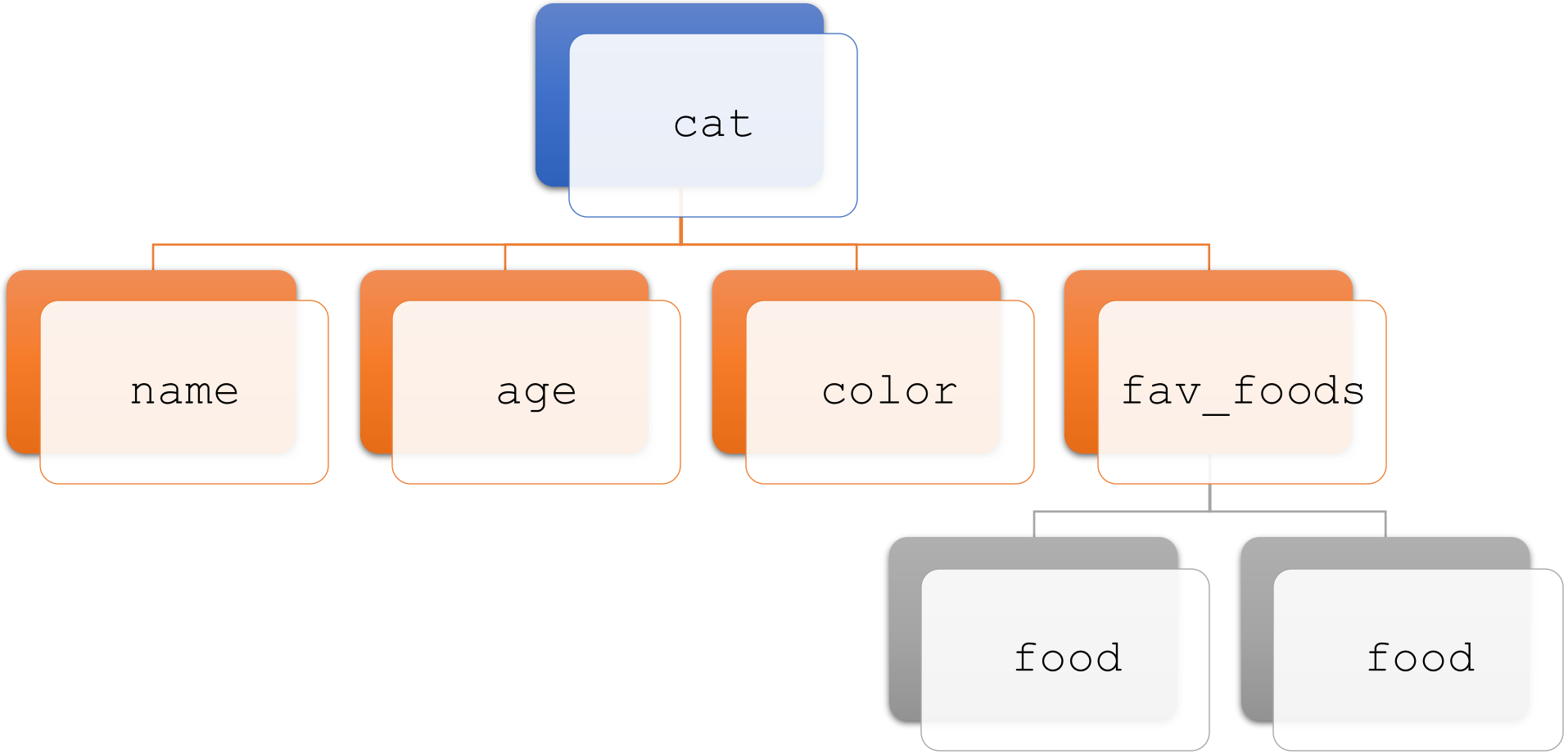
OPTION 1:

```
<cat id="1">
    <name>Salsa</name>
    <age>5</age>
    <color>black</color>
  </cat>
```

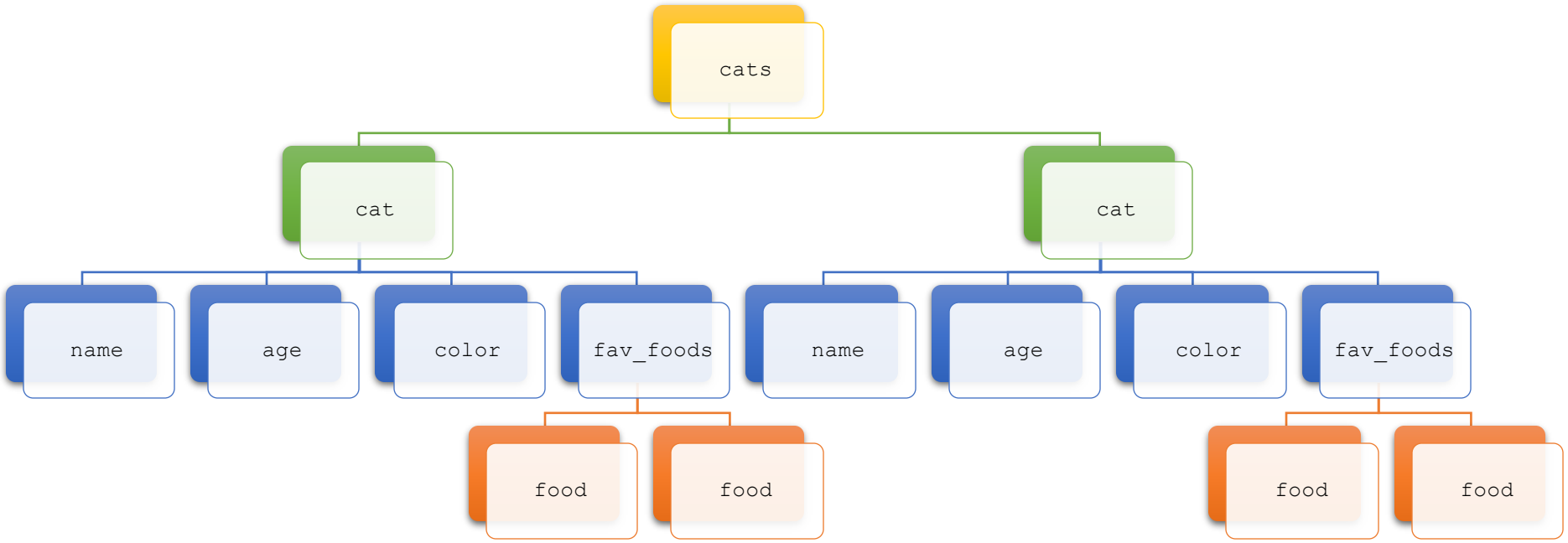
OPTION 2:

```
<cat>
    <id>1</id>
    <name>Salsa</name>
    <age>5</age>
    <color>black</color>
</cat>
```

Organizing Elements

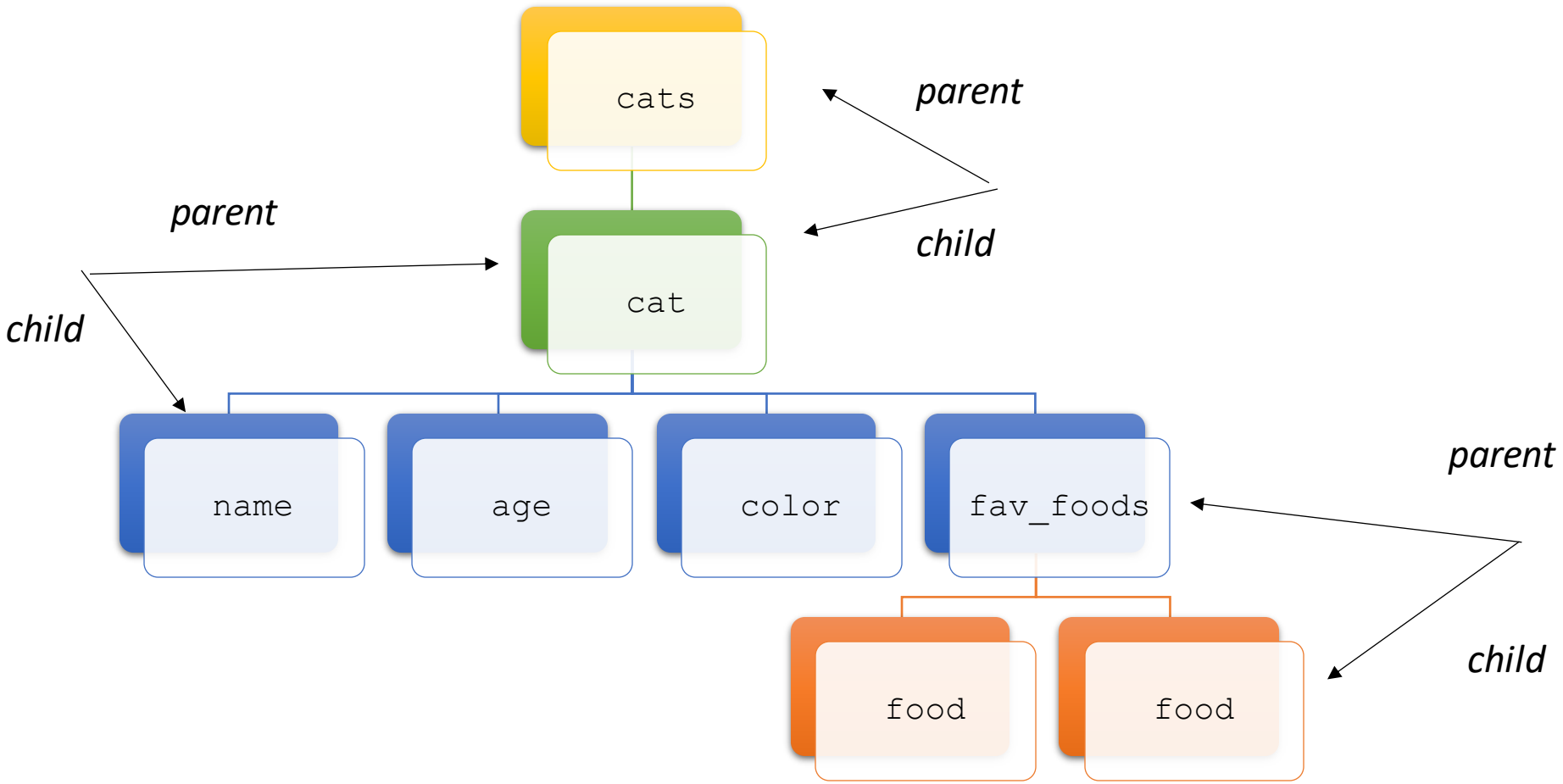


Organizing Elements

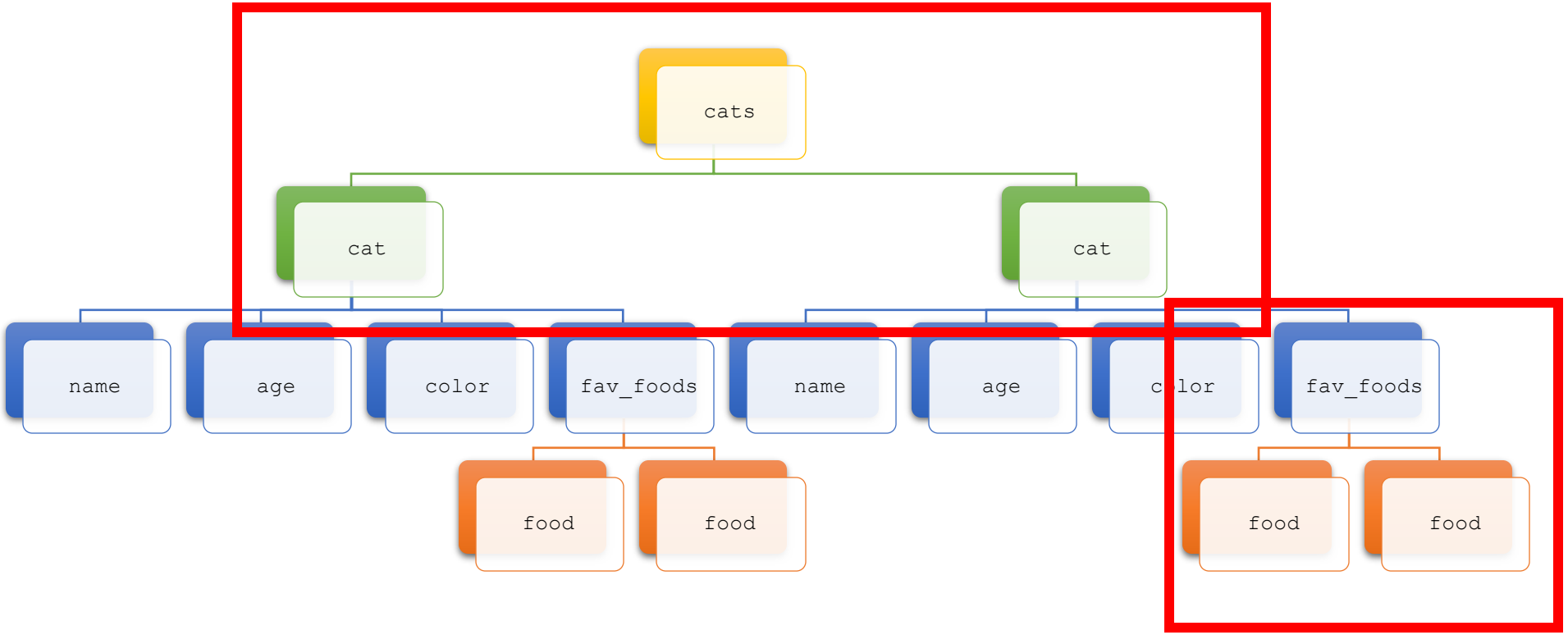



```
<cats>
  <cat id="1">
    <name>Salsa</name>
    <age>5</age>
    <color>black</color>
    <fav_foods>
        <food>Meow Mix</food>
        <food>Fancy Feast</food>
    <fav_foods>
  </cat>
  <cat id="2">
    <name>Gumbo</name>
    <age>2</age>
    <color>orange</color>
    <fav_foods>
        <food>Friskies Seafood</food>
        <food>Steals cheese!</food>
    </fav_foods>
  </cat>
</cats>
```

Types of Relationships



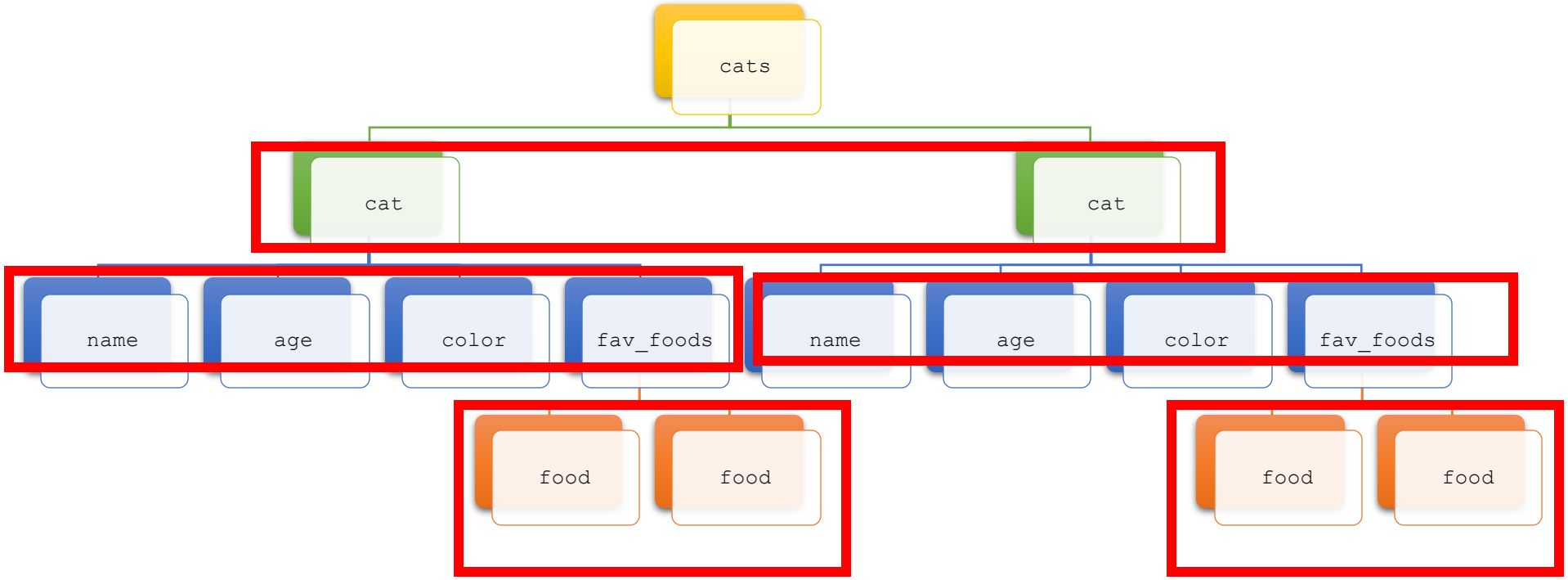
Types of Relationships




```
<cats>
  <cat id="1">
    <name>Salsa</name>
    <age>5</age>
    <color>black</color>
    <fav_foods>
        <food>Meow Mix</food>
        <food>Fancy Feast</food>
    <fav_foods>
  </cat>
  <cat id="2">
    <name>Gumbo</name>
    <age>2</age>
    <color>orange</color>
    <fav_foods>
        <food>Friskies Seafood</food>
        <food>Steals cheese!</food>
    </fav_foods>
  </cat>
</cats>
```

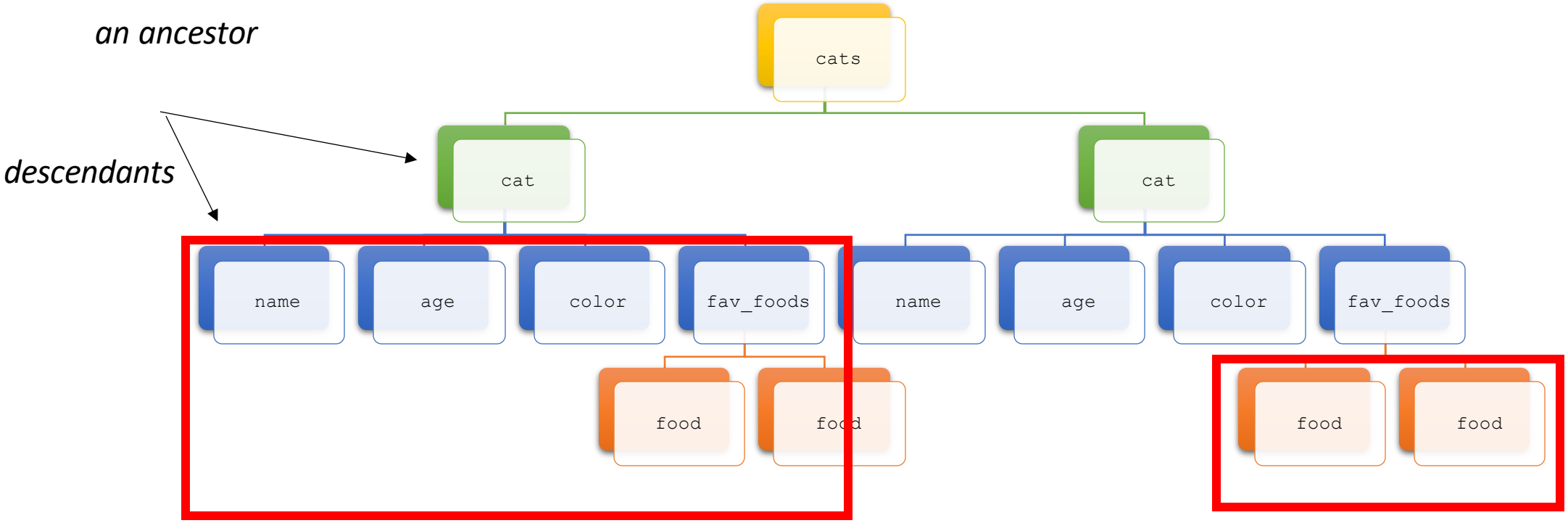
Parent – child Relationships

Types of Relationships



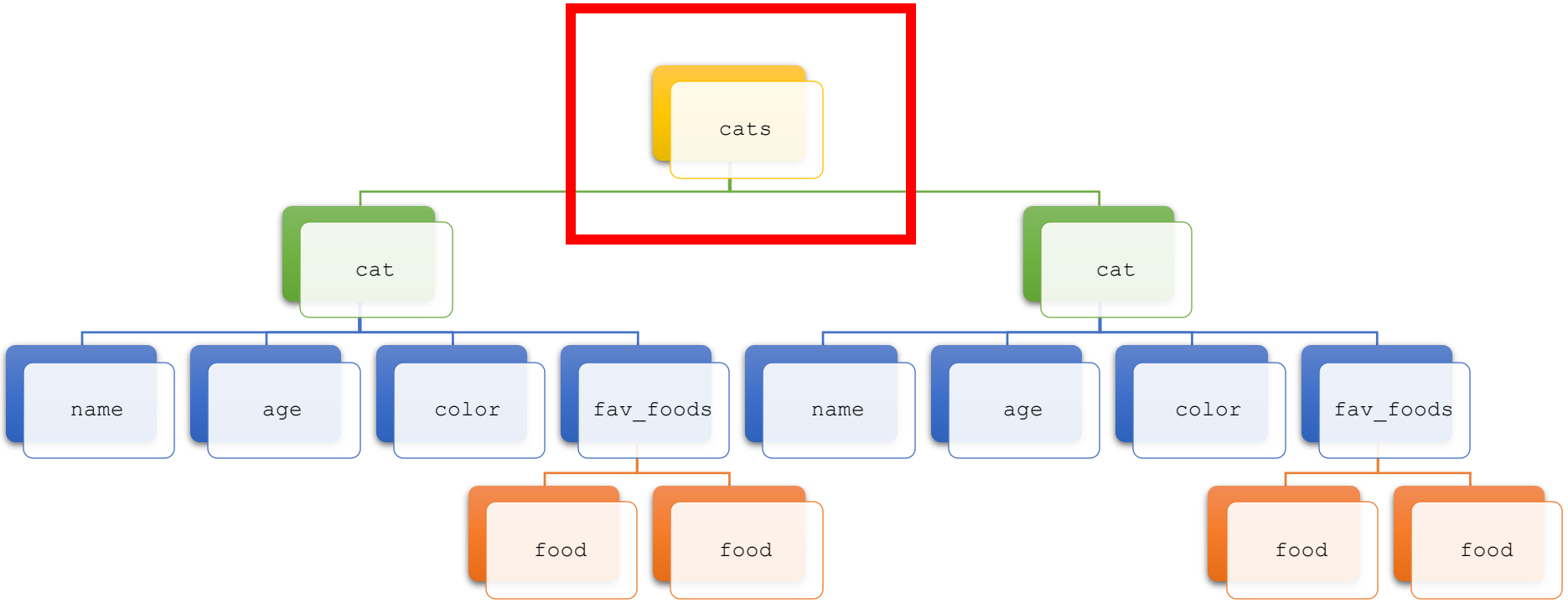
SIBLINGS share the same PARENT

Types of Relationships



All of these Descendants share the same ancestor

Types of Relationships



The root element is the highest level element in the tree.
Everything descends from the root element.

XML Schemas

Enforces:

- Acceptable data types (number, text, date, etc)
- The number of times an element can occur.
- What element names are acceptable.
- Allowable children for a particular parent.

Promotes consistency across records and documents.

VALIDATION

- Validation means XML conforms to schema.
- Languages have libraries and packages for validating programmatically. (Free)
- Free and paid tools also exist. Milage may vary:
  - Oxygen XML editor (paid but academic discount)
  - Various tools that may be free (https://www.freeformatter.com/xml-validator-xsd.html)

"WELL-FORMED"

- To be well-formed simply means that it follows the syntactical rules of XML.

- A document can be well-formed but still fail to be schema-valid.

XML Schemas (XSD)

- XSD: XML Schema Definition file
- XML document files can reference a schema.
- The schema can contain both rules as well as documentation for understanding the elements.

Simple & Complex Types

- Simple types are not comprised of multiple other elements:

```
<xs:element name="color" type="xs:string"/>
```

Simple & Complex Types

- Complex Types are containers for other elements and can implement the parent-child relationship:

```
<xs:element name="cat">
    <xs:complexType mixed="true">
```

Sequence

- Indicates a series of elements.
- Order and type matters.

```
<xs:element name="personinfo">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="firstname" type="xs:string"/>
   <xs:element name="lastname" type="xs:string"/>
   <xs:element name="address" type="xs:string"/>
   <xs:element name="city" type="xs:string"/>
   <xs:element name="country" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Super Simple Schema Example

namespace

cats-demo.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.monicaihli.com/lecture/cats-demo.xsd"
    elementFormDefault="qualified"
    xmlns="http://www.monicaihli.com/lecture/cats.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="cat">
        <xs:complexType mixed="true">
            <xs:sequence>
                <xs:element name="name" type="xs:string"/>
                <xs:element name="age" type="xs:integer"/>
                <xs:element name="color" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

parent element

child elements

Super Simple Schema Example

cats-demo.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema
targetNamespace="http://www.monicaihli.com/lecture/cats-
demo.xsd"

    elementFormDefault="qualified"

    xmlns="http://www.monicaihli.com/lecture/cats.xsd"

    xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="cat">

        <xs:complexType mixed="true">

            <xs:sequence>

                <xs:element name="name" type="xs:string"/>

                <xs:element name="age" type="xs:integer"/>

                <xs:element name="color" type="xs:string"/>

            </xs:sequence>

        </xs:complexType>

    </xs:element>

</xs:schema>
```

cats-demo.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<cat
  xmlns="http://www.monicaihli.com/lecture
  /cats.xsd" >

    <name>Salsa</name>

    <age>5</age>

    <color>black</color>

</cat>
```

Schemas In Practice

- It's up to you how strict you want to be.
- There is a trade-off for strictness: Data quality vs. uptake & adoption.
- Schemas are not content standards.
- Even the best standards & schemas leave room for ambiguities that you maybe did not anticipate.