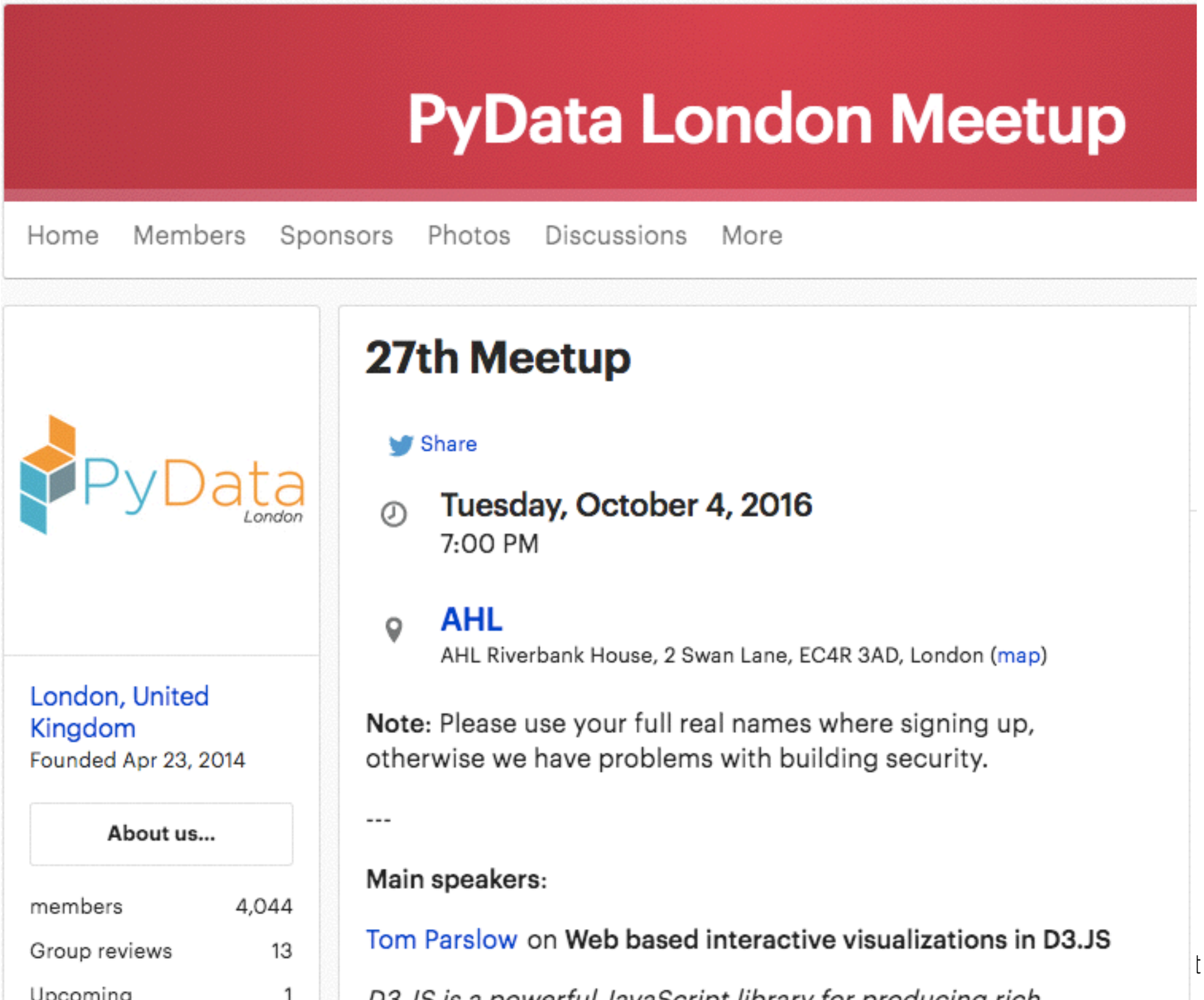
Introduction to Handling Data Part 2

Week 2: Computer Programming for Data Scientists (7CCSMCMP)

3 October 2016

Announcements

http://www.meetup.com/PyData-London-Meetup/events/234100340/



CSV File Structure

tabular data - 1 CSV file represents 1 table

rows in the table - **lines** in the file

columns in a row - **fields** (values) separated by commas* in each line

***delimiter** characters - separate values, typically commas (,), pipes (|), tabs (	)

columns names - first line (or lines) are **headers** in the file

quote characters - surround a field with quotes if they have special characters

Table

produce	quantity	price
apples	2	1.28
avocados		0.99
lemons, small	10	3.99
	2	2.30

File

```
produce,quantity,price
apples,2,1.28
avocados,,0.99
“lemons, small”,10,3.99
,2,2.380
```

*Use Python’s (built-in) **`csv`** module to parse and write CSV files*

Example **Data set - Mother's Country of Birth (2014)**

Source: http://www.ons.gov.uk/ons/rel/vsob1/parents--country-of-birth--england-and-wales/index.html

Available as Excel File (.XLS) - Exported as **CSV**

	A	B	C	D	E	F	G	H	I	J	K	L	M
1					Mothers born outside United Kingdom								
2							European Union						
3	Code	Area	All Live Births	Mothers Born within United Kingdom	Total	Percentage of live births to non-UK born mothers	Total	New EU	Rest of Europe (non EU)	Middle East and Asia	Africa	Rest of World	
4													
5	E09000002	Barking and Dagenham	3,569	1,288	2,280	63.9	537	476	149	675	852	67	
6	E09000003	Barnet	5,244	2,173	3,071	58.6	1,096	839	280	930	556	209	
7	E09000004	Bexley	3,037	2,072	964	31.7	235	166	38	210	432	49	
8	E09000005	Brent	5,078	1,292	3,786	74.6	1,058	807	116	1,593	745	274	
9	E09000006	Bromley	4,086	2,859	1,227	30.0	407	244	108	280	283	149	
10	E09000007	Camden	2,700	990	1,710	63.3	397	107	155	536	342	280	
11	E09000008	Croydon	5,645	2,670	2,973	52.7	759	518	130	913	870	301	
12	E09000009	Ealing	5,474	1,581	3,892	71.1	1,147	884	142	1,827	568	208	
13	E09000010	Enfield	4,824	1,895	2,928	60.7	831	679	616	411	873	197	
14	E09000011	Greenwich	4,368	1,803	2,564	58.7	624	415	134	643	980	183	
15	E09000012	Hackney and City of London											
16	E09000013	Hammersmith and Fulham	2,440	1,002	1,438	58.9	485	154	92	338	290	233	
17	E09000014	Haringey	4,006	1,442	2,564	64.0	958	674	288	415	586	317	
18	E09000015	Harrow	3,525	1,061	2,464	69.9	799	710	70	1,221	314	60	
19	E09000016	Havering	3,150	2,259	891	28.3	293	237	63	231	262	42	
20	E09000017	Hillingdon	4,423	1,964	2,459	55.6	562	430	73	1,294	448	82	
21	E09000018	Hounslow	4,245	1,414	2,831	66.7	768	608	109	1,394	427	133	
22	E09000019	Islington	2,879	1,357	1,522	52.9	428	136	165	272	404	253	
23	E09000020	Kensington and Chelsea	1,821	533	1,288	70.7	404	76	133	274	213	264	
24	E09000021	Kingston upon Thames	2,247	1,191	1,056	47.0	326	191	74	396	152	108	
25	E09000022	Lambeth	4,528	2,034	2,493	55.1	757	311	86	298	770	582	
26	E09000023	Lewisham	4,759	2,156	2,602	54.7	598	349	129	542	940	393	
27	E09000024	Merton	3,274	1,356	1,918	58.6	638	436	99	636	346	199	
28	E09000025	Newham	6,023	1,401	4,621	76.7	1,064	909	143	2,347	907	160	
29	E09000026	Redbridge	4,678	1,654	3,024	64.6	700	572	122	1,720	381	101	
30	E09000027	Richmond upon Thames	2,589	1,583	1,006	38.9	391	156	90	223	113	189	
31	E09000028	Southwark	4,647	1,858	2,789	60.0	510	215	127	476	1,212	464	
32	E09000029	Sutton	2,751	1,645	1,105	40.2	347	242	70	396	221	71	
33	E09000030	Tower Hamlets	4,619	1,634	2,985	64.6	396	170	89	1,964	359	177	
34	E09000031	Waltham Forest	4,618	1,746	2,872	62.2	1,020	824	227	846	588	191	
35	E09000032	Wandsworth	5,110	2,543	2,567	50.2	855	337	125	604	511	472	
36	E09000033	Westminster	2,604	706	1,898	76.4	424	109	183	632	329	330	
37													

Example **Data set - Mother's Country of Birth (2014)**

Source: http://www.ons.gov.uk/ons/rel/vsob1/parents--country-of-birth--england-and-wales/index.html

Excel File

	A	B	C	D	E	F	G	H	I	J	K	L	M
1					Mothers born outside United Kingdom								
2							European Union						
3	Code	Area	All Live Births	Mothers Born within United Kingdom	Total	Percentage of live births to non-UK born mothers	Total	New EU	Rest of Europe (non EU)	Middle East and Asia	Africa	Rest of World	
4													
5	E09000002	Barking and Dagenham	3,569	1,288	2,280	63.9	537	476	149	675	852	67	
6	E09000003	Barnet	5,244	2,173	3,071	58.6	1,096	839	280	930	556	209	
7	E09000004	Bexley	3,037	2,072	964	31.7	235	166	38	210	432	49	
8	E09000005	Brent	5,078	1,292	3,786	74.6	1,058	807	116	1,593	745	274	
9	E09000006	Bromley	4,086	2,859	1,227	30.0	407	244	108	280	283	149	
10	E09000007	Camden	2,700	990	1,710	63.3	397	107	155	536	342	280	
11	E09000008	Croydon	5,645	2,670	2,973	52.7	759	518	130	913	870	301	
12	E09000009	Ealing	5,474	1,581	3,892	71.1	1,147	884	142	1,827	568	208	
13	E09000010	Enfield	4,824	1,895	2,928	60.7	831	679	616	411	873	197	
14	E09000011	Greenwich	4,368	1,803	2,564	58.7	624	415	134	643	980	183	

CSV File

```
,,,,Mothers born outside United Kingdom,,,,,,,,
,,,,,,European Union,,,,,,,
Code,Area,All Live Births,Mothers Born within United Kingdom,Total,Percentage of live births to non-UK
born mothers,Total,New EU,Rest of Europe (non EU),Middle East and Asia,Africa,Rest of World ,,
,,,,,,,,,,,,,
E09000002,Barking and Dagenham,"3,796","1,411","2,383",62.8,595,527,123,692,915,58,,
E09000003,Barnet,"5,187","2,216","2,971",57.3,977,738,260,953,597,184,,
E09000004,Bexley,"2,959","2,019",940,31.8,191,137,49,181,468,51,,
E09000005,Brent,"5,170","1,243","3,926",75.9,"1,060",847,104,"1,658",795,309,,
E09000006,Bromley,"3,899","2,694","1,205",30.9,400,232,88,270,271,176,,
E09000007,Camden,"2,766","1,081","1,685",60.9,401,115,145,545,314,280,,
E09000008,Croydon,"5,605","2,713","2,892",51.6,703,507,139,835,897,318,,
E09000009,Ealing,"5,395","1,607","3,788",70.2,"1,136",839,142,"1,708",611,191,,
E09000010,Enfield,"4,908","1,938","2,970",60.5,799,636,652,412,900,207,,
E09000011,Greenwich,"4,442","1,921","2,521",56.8,544,371,129,648,"1,035",165,,
```

CSV Files - Parsing with `csv.reader`

Python Doc: https://docs.python.org/2/library/csv.html#csv.reader

`csv.reader()` reads a CSV file into a **list of lists**

```
import csv # import the csv module

try:
    with open("data/mothers.csv", "r") as mothers_fd: # open a file context
        csv_data = csv.reader(mothers_fd)
        mothers = list(csv_data) # converts the *iterator* into a list
except IOError as ioe:
    print("IOError: " + str(ioe))
```

List of lists representation for: **n** columns, **m** rows

[[*<row 1 column 1>***,** *<row 1 column 2>, … ,* *<row 1 column **n***>* **],**
[*<row 2 column 1>***,** *<row 2 column 2>, … ,* *<row 2 column **n***>* **],**
…
[*<row m column 1>***,** *<row m, column 2>, … ,* *<row **m** column **n***>* **]]**

CSV Files - Parsing with `csv.reader`

Python Doc: https://docs.python.org/2/library/csv.html#csv.reader

`csv.reader()` reads a CSV file into a **list of lists**

```
import csv # import the csv module

try:
    with open("data/mothers.csv", "r") as mothers_fd: # open a file context
        csv_data = csv.reader(mothers_fd)
        mothers = list(csv_data) # converts the *iterator* into a list
except IOError as ioe:
    print("IOError: " + str(ioe))
```

```
[['', '', '', '', 'Mothers born outside United Kingdom', '', '', '', '', '',
  '', '', '', ''],
 ['', '', '', '', '', '', 'European Union', '', '', '', '', '', '', ''],
 ['Code', 'Area', 'All Live Births', 'Mothers Born within United Kingdom',
  'Total', 'Percentage of live births to non-UK born mothers', 'Total',
  'New EU', 'Rest of Europe (non EU)', 'Middle East and Asia', 'Africa',
   'Rest of World ', '', ''],
  ['', '', '', '', '', '', '', '', '', '', '', '', '', ''],
  ['E09000002', 'Barking and Dagenham', '3,796', '1,411', '2,383', '62.8',
   '595', '527', '123', '692', '915', '58', '', '']…
```

CSV Files - Parsing with `csv.reader`

`mothers[4][1]`

CSV

```
,,,,Mothers born outside United Kingdom,,,,,,,,
,,,,,,European Union,,,,,,,
Code,Area,All Live Births,Mothers Born within United Kingdom,Total,Percentage of live births to non-UK born mothers,Total,New EU,Rest of Europe (non EU),Middle East and Asia,Africa,Rest of World ,,
,,,,,,,,,,,,,,
E09000002,Barking and Dagenham,"3,796","1,411","2,383",62.8,595,527,123,692,915,58,,
```

`mothers` variable (***list of list*** representation)

```
[['', '', '', '', 'Mothers born outside United Kingdom', '', '', '', '', '',
  '', '', '', ''],
 ['', '', '', '', '', '', 'European Union', '', '', '', '', '', '', ''],
 ['Code', 'Area', 'All Live Births', 'Mothers Born within United Kingdom’,
  'Total', 'Percentage of live births to non-UK born mothers', 'Total',
  'New EU', 'Rest of Europe (non EU)', 'Middle East and Asia', 'Africa',
   'Rest of World ', '', ''],
  ['', '', '', '', '', '', '', '', '', '', '', '', '', ''],
  ['E09000002', 'Barking and Dagenham', '3,796', '1,411', '2,383', ’62.8',
   '595', '527', '123', '692', '915', '58', '', ‘’]…
```

CSV Files - Parsing with `csv.reader`

`mothers[0][4]`

CSV

```
,,,,Mothers born outside United Kingdom,,,,,,,,
,,,,,,European Union,,,,,,,
Code,Area,All Live Births,Mothers Born within United Kingdom,Total,Percentage of live
births to non-UK born mothers,Total,New EU,Rest of Europe (non EU),Middle East and
Asia,Africa,Rest of World ,,
,,,,,,,,,,,,,,
E09000002,Barking and Dagenham,"3,796","1,411","2,383",62.8,595,527,123,692,915,58,,
```

`mothers` variable (***list of list*** representation)

```
[['', '', '', '', 'Mothers born outside United Kingdom', '', '', '', '', '',
  '', '', '', ''],
 ['', '', '', '', '', '', 'European Union', '', '', '', '', '', '', ''],
 ['Code', 'Area', 'All Live Births', 'Mothers Born within United Kingdom’,
  'Total', 'Percentage of live births to non-UK born mothers', 'Total',
  'New EU', 'Rest of Europe (non EU)', 'Middle East and Asia', 'Africa',
   'Rest of World ', '', ''],
  ['', '', '', '', '', '', '', '', '', '', '', '', '', ''],
  ['E09000002', 'Barking and Dagenham', '3,796', '1,411', '2,383', ’62.8',
   '595', '527', '123', '692', '915', '58', '', ‘’]…
```

CSV Files - Parsing with `csv.reader`

`mothers[3]`

CSV

```
,,,,Mothers born outside United Kingdom,,,,,,,,
,,,,,,European Union,,,,,,,
Code,Area,All Live Births,Mothers Born within United Kingdom,Total,Percentage of live births to non-UK born mothers,Total,New EU,Rest of Europe (non EU),Middle East and Asia,Africa,Rest of World ,,
,,,,,,,,,,,,,
E09000002,Barking and Dagenham,"3,796","1,411","2,383",62.8,595,527,123,692,915,58,,
```

`mothers` variable (***list of list*** representation)

```
[['', '', '', '', 'Mothers born outside United Kingdom', '', '', '', '', '',
  '', '', '', ''],
 ['', '', '', '', '', '', 'European Union', '', '', '', '', '', '', ''],
 ['Code', 'Area', 'All Live Births', 'Mothers Born within United Kingdom’,
  'Total', 'Percentage of live births to non-UK born mothers', 'Total',
  'New EU', 'Rest of Europe (non EU)', 'Middle East and Asia', 'Africa',
   'Rest of World ', '', ''],
  ['', '', '', '', '', '', '', '', '', '', '', '', '', ''],
  ['E09000002', 'Barking and Dagenham', '3,796', '1,411', '2,383', ’62.8',
   '595', '527', '123', '692', '915', '58', '', ‘’]…
```

CSV Files - Parsing with `csv.DictReader()`

Python Doc: https://docs.python.org/2/library/csv.html#csv.DictReader

- Uses a list of *field names* (i.e. the *header* row) to reference columns in a row
- Define the field names with an argument to the DictReader (i.e. `fieldnames`)
- Order of field names should be the order of the columns in the header
- `csv.DictReader()` reads a CSV file into a **list of dicts**

```
import csv # import the csv module

try:
    with open("data/mothers.csv", "r") as mothers_fd: # open a file context
        csv_data = csv.DictReader(mothers_fd, fieldnames=['Code','Area'])
        mothers = list(csv_data) # converts the *iterator* into a list
except IOError as ioe:
    print("IOError: " + str(ioe))
```

List of lists representation for: **m** rows, columns referenced by `fieldnames`

[{“<field name 1>”**:** *<row 1 column 1>*, “<field name 2>”**:** *<row 1 column 1>, …* **},**
{“<field name 1>”**:** *<row 2 column 1>*, “<field name 2>”**:** *<row 2 column 1>, …* **},**
…
{“<field name 1>”**:** *<row m column 1>*, “<field name 2>”**:** *<row m column 1>, …* **}]**

CSV Files - Parsing with `csv.DictReader()`

`mothers[4]["Code"]`

csv

```
,,,,Mothers born outside United Kingdom,,,,,,,,
,,,,,,European Union,,,,,,,
Code,Area,All Live Births,Mothers Born within United Kingdom,Total,Percentage of live births to non-UK born mothers,Total,New EU,Rest of Europe (non EU),Middle East and Asia,Africa,Rest of World ,,
,,,,,,,,,,,,,
E09000002,Barking and Dagenham,"3,796","1,411","2,383",62.8,595,527,123,692,915,58,,
```

`mothers` variable (***list of dicts*** representation)

```
[{'Area': '', 'Code': '', …},
 {'Area': '', 'Code': '', …},
 {'Area': 'Area', 'Code': 'Code', …},
 {'Area': '','Code': '', …},
 {'Area': 'Barking and Dagenham','Code': 'E09000002', …}, …]
```

CSV Files - Cleaning Raw Data

- many empty rows (i.e. records) in the this raw data
- need to clean the data by removing the empty records
- identify an empty record by whether it is missing the “Code” and remove it
- whether or not you decide to remove records and rows depends on the dataset and the task at hand (i.e. you may want missing values)

```
[{'Area': '', 'Code': ‘’, …},
 {'Area': '', 'Code': ‘’, …},
 {'Area': ‘Area', 'Code': ‘Code’, …},
 {'Area': '','Code': ‘’, …},
 {‘Area': 'Barking and Dagenham','Code': ‘E09000002’, …},
 {‘Area': ‘Barnet','Code': ‘E09000003’, …},
 {‘Area': 'Bexley','Code': ‘E09000004’, …},
 {‘Area': 'Brent','Code': ‘E09000005’, …},
…]
```

CSV Files - Cleaning Raw Data

- Copy to a *new* list (`mothers2`) the records that you want to keep

```
# show the number of all records
print "Number of records: %d" % len(mothers)

mothers2 = []
for rec in mothers:
    if len(rec['Code']) > 0: # test the length of the field value
        mothers2.append(rec)

# show the number of remaining "cleaned" records
print "Number of clean records: %d" % len(mothers2)

# print out the "Code" and "Area, again
for rec in mothers2:
    print "%s %s" % (rec["Code"], rec["Area"])
```

```
Number of records: 57
Number of clean records: 48
Code Area
E09000002 Barking and Dagenham
E09000003 Barnet
E09000004 Bexley
E09000005 Brent
E09000006 Bromley
E09000007 Camden
E09000008 Croydon
```

CSV Files - Writing out CSV files

- use the `csv.DictWriter` to write-back the cleaned dataset
- need to define the `fieldnames` and specify to ignore extra fields (`extraction`)
- use `writeheader()` write the header for the CSV file
- `writerows()` outputs all of the records

```
import csv

with open('mothers_areas.csv', 'w') as csvfile:
    # define fieldnames again
    writer = csv.DictWriter(csvfile, fieldnames=['Code', 'Area'],
                            extrasaction='ignore')

    writer.writeheader() # writes the CSV column header
    writer.writerows(mothers2) # all of the records,
                               # ignore any non-specified columns

# look at mothers_areas.csv in File Manager
```

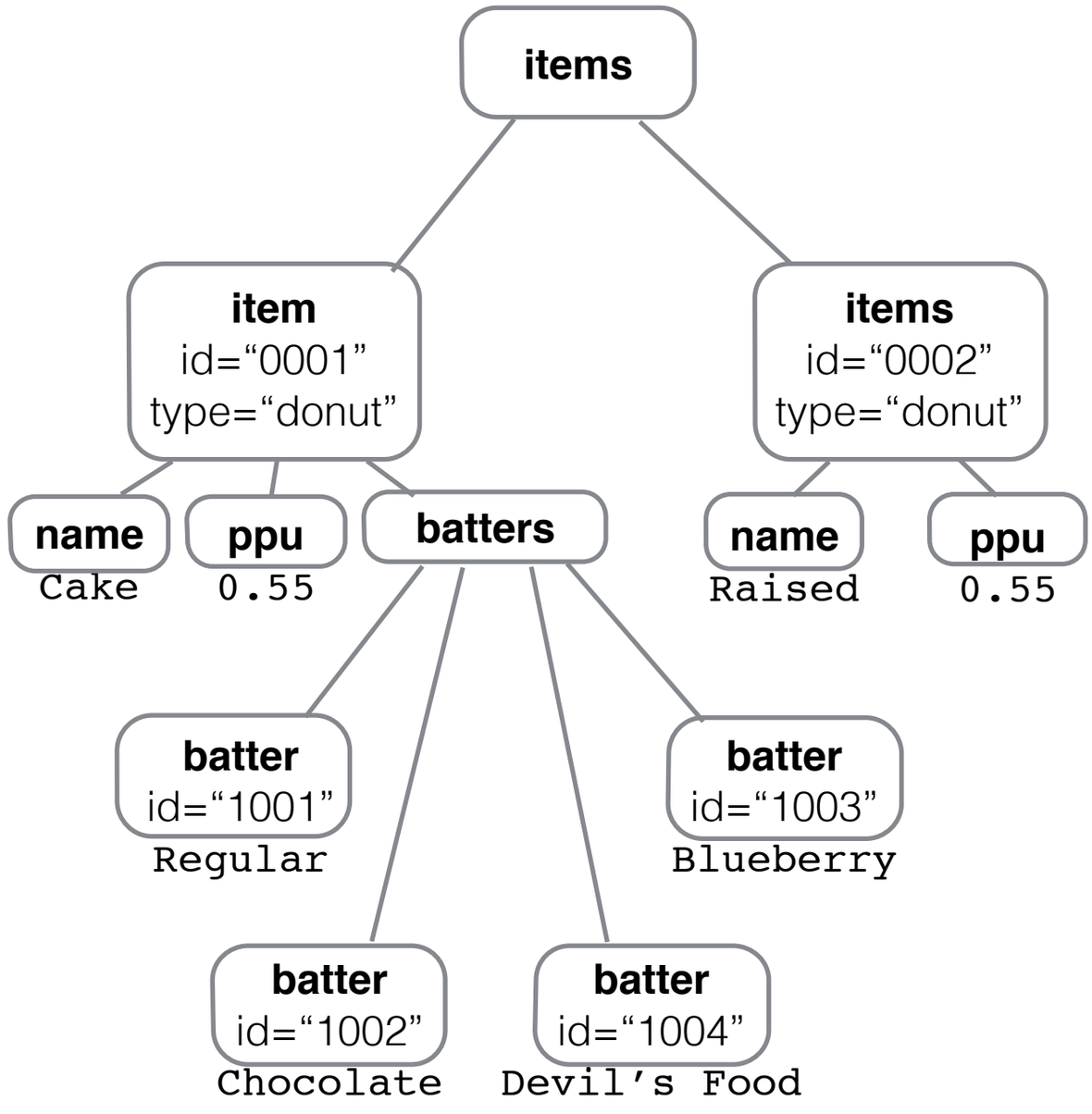
- Output CSV File: `mothers_areas.csv`

```
1 Code,Area
2 Code,Area
3 E09000002,Barking and Dagenham
4 E09000003,Barnet
5 E09000004,Bexley
6 E09000005,Brent
7 E09000006,Bromley
8 E09000007,Camden
9 E09000008,Croydon
```

XML Files - XML file structure

- XML files can represent *hierarchical* data, often visualized as a *tree* structure
- HTML (i.e. the structure of web-pages) is a form of XML

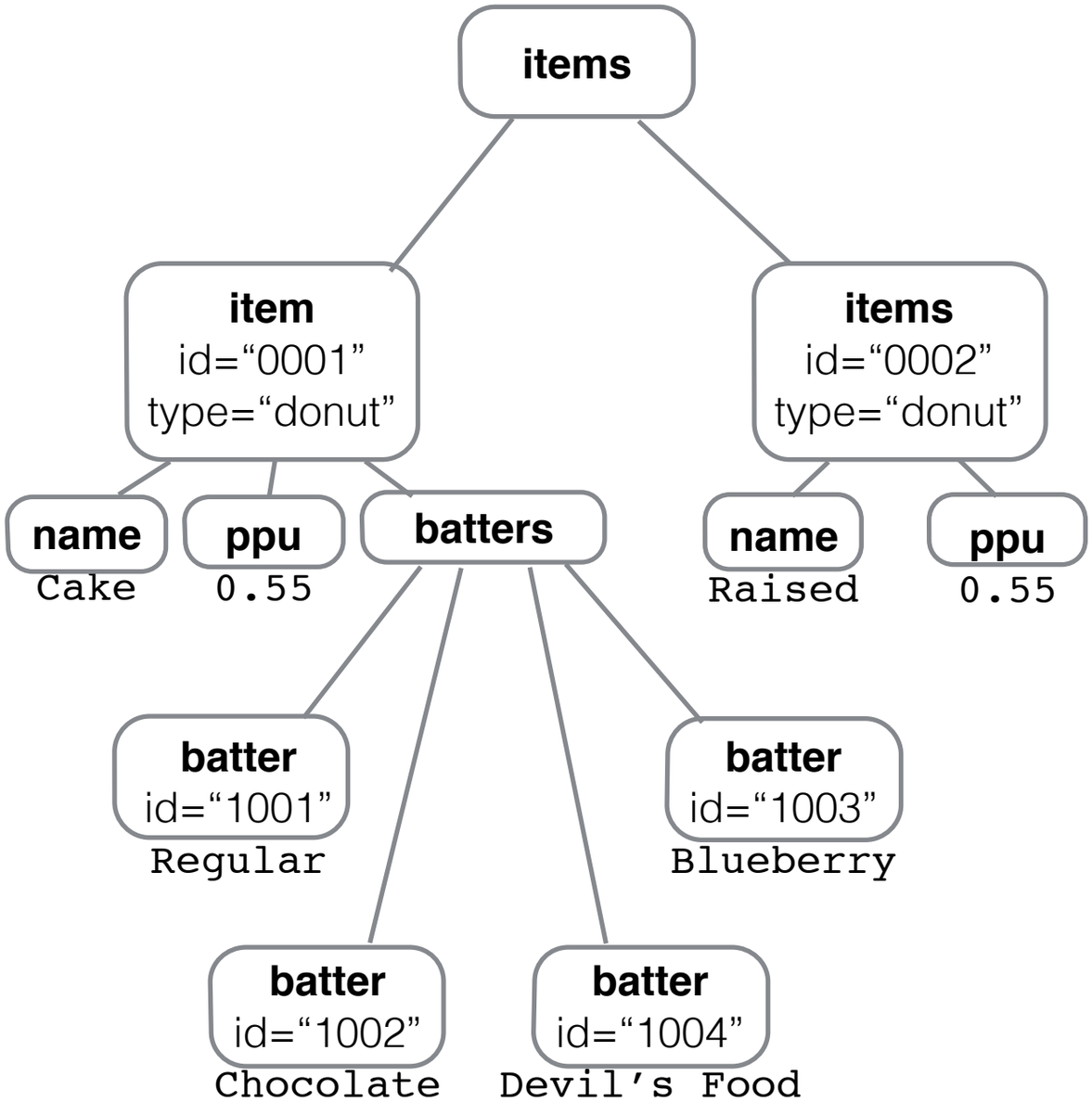
```
<items>
  <item id="0001" type="donut">
    <name>Cake</name>
    <ppu>0.55</ppu>
    <batters>
      <batter id="1001">Regular</batter>
      <batter id="1002">Chocolate</batter>
      <batter id="1003">Blueberry</batter>
      <batter id="1004">Devil's Food</batter>
    </batters>
  </item>
  <item id="0002" type="donut">
    <name>Raised</name>
    <ppu>0.55</ppu>
  </item>
</items>
```



XML Files - XML file structure

- XML structure is a tree structure of **Elements**
- Each Element is a *flexible container*, which parent-child relationships
- A ***parent*** element can contain child ***elements***
- Top-most parent element is the ***root*** of the tree

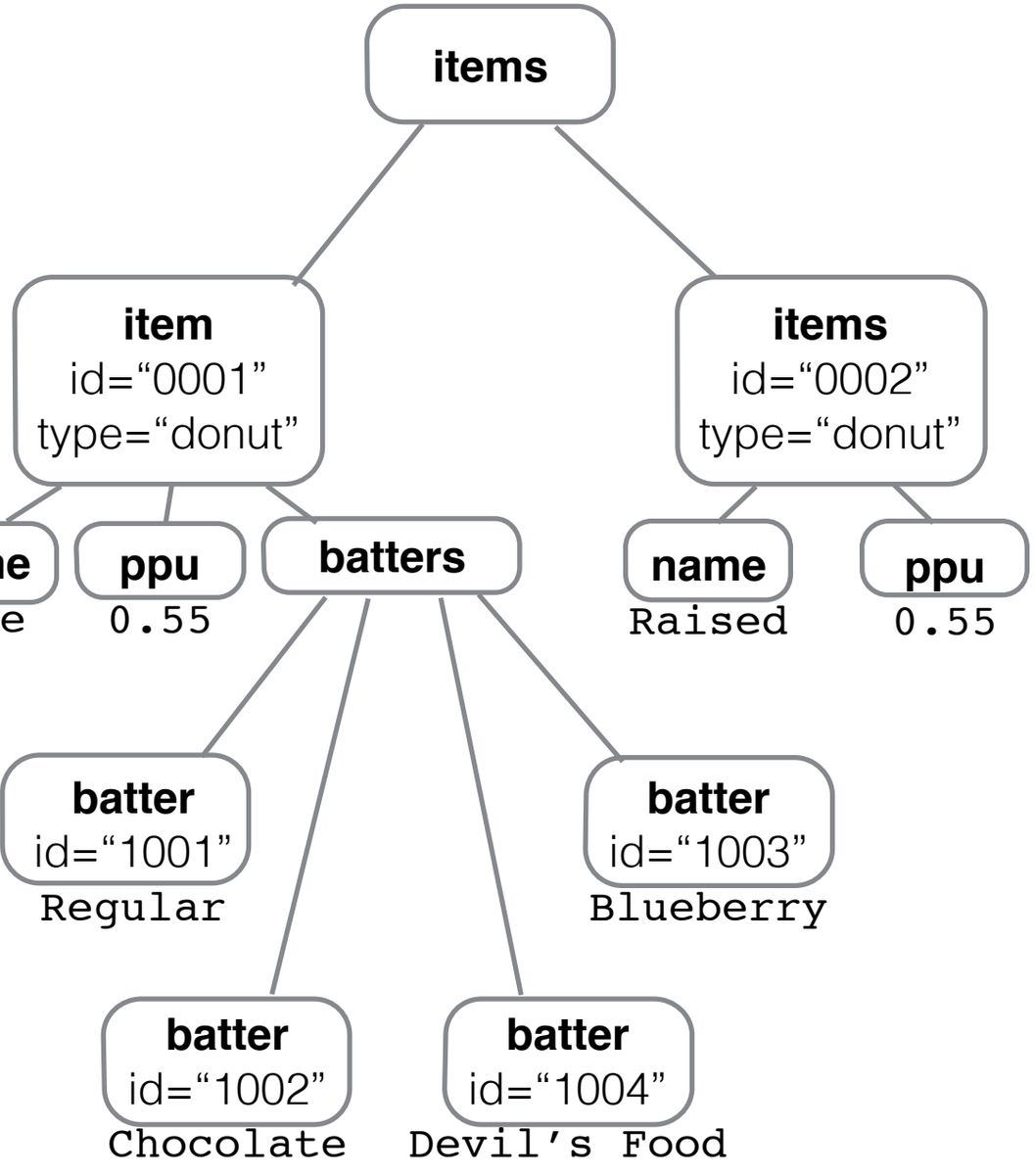
```
<items>
  <item id="0001" type="donut">
    <name>Cake</name>
    <ppu>0.55</ppu>
    <batters>
      <batter id="1001">Regular</batter>
      <batter id="1002">Chocolate</batter>
      <batter id="1003">Blueberry</batter>
      <batter id="1004">Devil's Food</batter>
    </batters>
  </item>
  <item id="0002" type="donut">
    <name>Raised</name>
    <ppu>0.55</ppu>
  </item>
</items>
```



XML Files - XML file structure

- Each **Element** has:
  - a *tag* - a *string* with opening and closing elements
  - *attributes - (i.e.* `id="0001"`*)*
  - *text* - the containing text (optional)
  - a number of *child* elements - other **Elements**

```
<items>
  <item id="0001" type="donut">
    <name>Cake</name>
    <ppu>0.55</ppu>
    <batters>
      <batter id="1001">Regular</batter>
      <batter id="1002">Chocolate</batter>
      <batter id="1003">Blueberry</batter>
      <batter id="1004">Devil's Food</batter>
    </batters>
  </item>
  <item id="0002" type="donut">
    <name>Raised</name>
    <ppu>0.55</ppu>
  </item>
</items>
```



XML Files - using `xml.etree.ElementTree` module

```
import xml.etree.ElementTree as et

try:
    tree = et.ElementTree(file='data/donuts-simple.xml')
    root = tree.getroot()
    for child in root:
        print("tag: '%s' attributes: %s text: '%s'" %
              (child.tag, child.attrib, child.text.strip()))
        for grchild in child:
            print("- tag: '%s' attributes: %s text: '%s'" %
                  (grchild.tag, grchild.attrib, grchild.text.strip()))
except Exception as e:
    print("Error %s" % e)
```

```
tag: 'item' attributes: {'type': 'donut', 'id': '0001'} text: ''
- tag: 'name' attributes: {} text: 'Cake'
- tag: 'ppu' attributes: {} text: '0.55'
- tag: 'batters' attributes: {} text: ''
tag: 'item' attributes: {'type': 'donut', 'id': '0002'} text: ''
- tag: 'name' attributes: {} text: 'Raised'
- tag: 'ppu' attributes: {} text: '0.55'
```

XML Files - using `xml.etree.ElementTree` module

- `xml.etree.ElementTree` API is "*a simple and lightweight XML processor*"
- Python Doc: https://docs.python.org/2/library/xml.etree.elementtree.html
- an `ElementTree` is a tree of `Element` objects
- each `Element` has the following fields:
  - `tag` - a string
  - `attributes` - a dictionary of attributes
  - `text` - a string (optional)
  - a number of child elements - a sequence of child `Element` objects

Navigating an ElementTree

- the `getroot()` function returns an Element object which is the root of the tree
- the `iter()` function returns a list of the children (`Element` objects)
- the `find( match_tag )` function returns an `Element` which is the first child (matching the specific `match_tag` string)
- the `findall( match_tag )` function returns all the children as `Element` objects

Used for *web-scraping* HTML pages

see Beautiful Soup - https://www.crummy.com/software/BeautifulSoup/bs4/doc/

XML Files - navigating entire tree

- using *recursion* to print the contents of an ElementTree of arbitrary *depth* and *breadth*

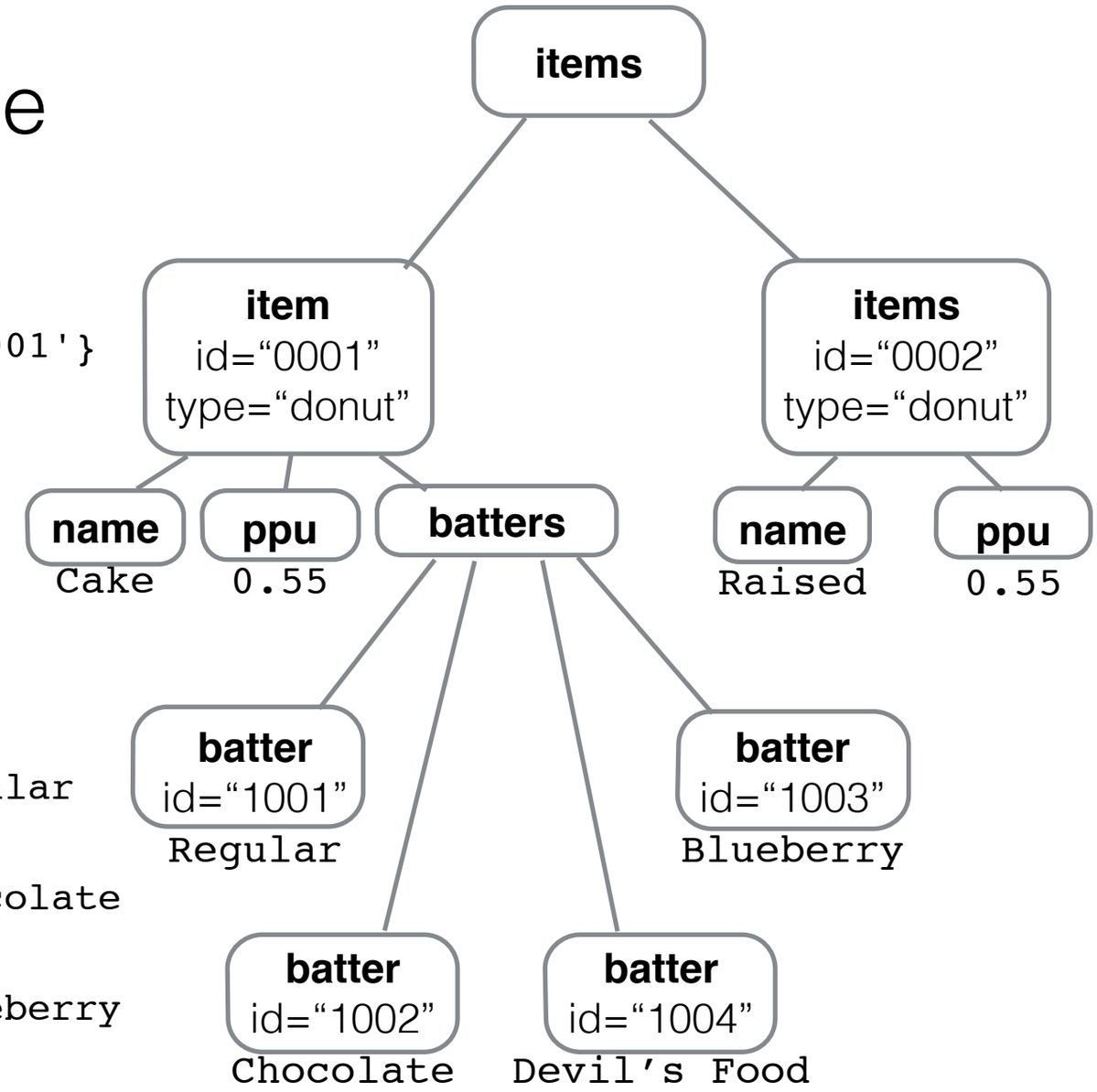
```
import xml.etree.ElementTree as et

 # recursive function
def display_child(child, level):
    print("(%d) tag: %s attributes: %s text: %s" %
          (level, child.tag, child.attrib, child.text.strip()))
    if(len(child) > 0):
        print("(%d) has %s children" % (level, len(child)))
        for grandchild in child:
            display_child(grandchild, level+1)
    else:
        print("(%d) has no children" % level)

# parse the tree recursively
tree = et.ElementTree(file='data/donuts-simple.xml')
root = tree.getroot()
display_child(root, 0)
```

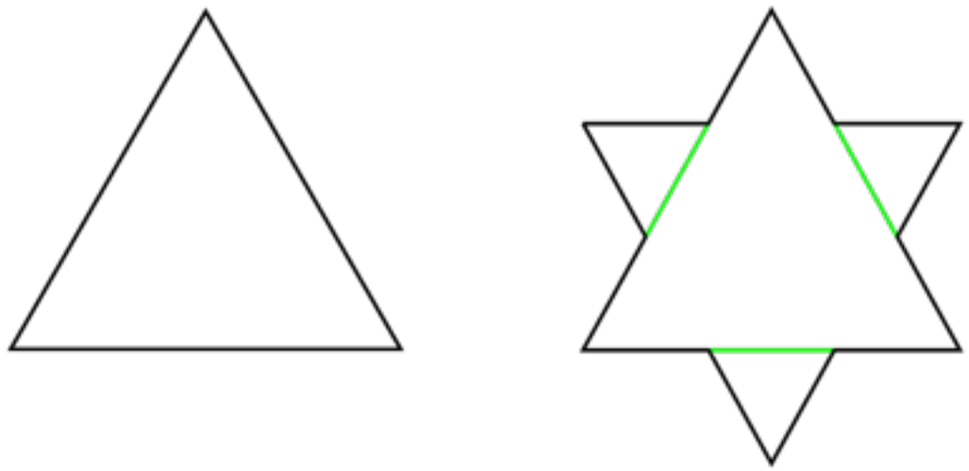
XML Files - navigating entire tree

```
(0) tag: items attributes: {} text:
(0) has 2 children
(1) tag: item attributes: {'type': 'donut', 'id': ‘0001'}
(1) has 3 children
(2) tag: name attributes: {} text: Cake
(2) has no children
(2) tag: ppu attributes: {} text: 0.55
(2) has no children
(2) tag: batters attributes: {} text:
(2) has 4 children
(3) tag: batter attributes: {'id': '1001'} text: Regular
(3) has no children
(3) tag: batter attributes: {'id': '1002'} text: Chocolate
(3) has no children
(3) tag: batter attributes: {'id': '1003'} text: Blueberry
(3) has no children
(3) tag: batter attributes: {'id': '1004'} text: Devil's Food
(3) has no children
(1) tag: item attributes: {'type': 'donut', 'id': '0002'} text:
(1) has 2 children
(2) tag: name attributes: {} text: Raised
(2) has no children
(2) tag: ppu attributes: {} text: 0.55
(2) has no children
```



Recursion - concept

- *recursion* is defining something in terms of itself
- many examples in natural and mathematics (i.e. fractals)



Koch's snowflake (wikipedia)

Recursion

Russian Stacking Dolls (Matroyshka dolls)

Recursion - classic examples

- classic examples of *recursion* in computer programming include:
  - the *power* function

$$x^y = \begin{cases} \text{if } y == 0,\ x^y = 1 \\ \text{if } y == 1,\ x^y = x \\ \text{otherwise, } x^y = x * x^{y-1} \end{cases}$$

  - the *factorial* function

$$N! = \begin{cases} \text{if } N == 1,\ N! = 1 \\ \text{otherwise, } N! = N * (N-1)! \end{cases}$$

- defining a recursive function requires:
  - the **stopping** condition - i.e. when to stop and how to stop
    - for *power*: $y == 0$ and $y == 1$
    - for *factorial*: $N == 1$
  - the **recursing** condition - i.e. when to keep going and how
    - for *power*: x^{y-1}
    - for *factorial*: $(N - 1)!$
- *be careful!* because it is easy to define and infinitely recurring function…

Recursion - *power* function using recursion

```
def power(x, y):
    if(y == 0):
        return 1
    elif(y == 1):
        return x
    else:
        return x * power(x, y-1)

# main
print("2^3 = %s" % power(2,3))
```

```
2^3 = 8
```

stopping conditions

recursing condition

Recursion - *factorial* function using recursion

```
def fact(n):
    if(n == 1):
        return 1
    else:
        return n * fact(n-1)

# use the factorial function
print("5! = %s" % fact(5))
```

stopping condition

recursing condition

```
5! = 120
```

Summary

CSV - tabular data format
- reading list of lists with csv.reader
- reading list of dictionaries with csv.DictReader
- cleaning data
- writing with csv.DictWriter

XML - hierarchical data format
- using ElementTree to parse tree
- parsing an XML tree recursively

Recursion - computer programing concept
- *power* function
- *factorial* function