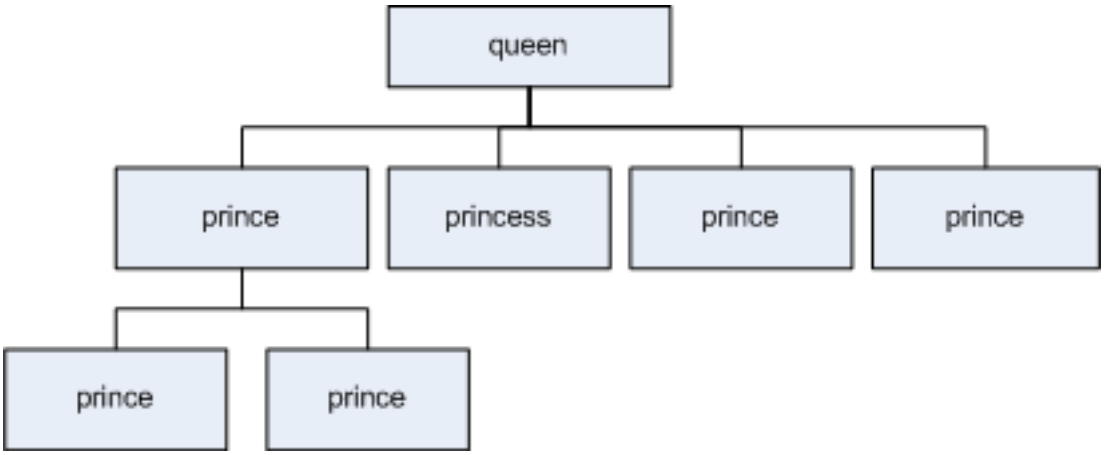
1. HTML markup is concerned with presentation in the browser for humans to view. Markup vocabulary is fixed. For XML, markup (elements and attributes) can be defined by users and their purpose is to specify the semantics of the data so that it is machine processable. Use HTML for data that will be viewed in browser by humans, formatted nicely. XML for applications such as data exchange, integration, export into other formats, where semantics is needed, etc.

2.

```
<?xml version="1.0" encoding="utf-8"?>
<queen title="Queen Elizabeth II" marriedTo="Philip, Duke of
Edinburgh">
    <!-- error: <prince title="Charles, Prince of Wales"
marriedTo="Lady Diana Spencer"> -->
    <prince title="Charles, Prince of Wales" marriedTo="Lady
Diana Spencer">
      <prince title="Prince William of Wales" />
      <prince title="Prince Henry of Wales" />
    </prince>
    <princess title="Anne, Princess Royal" />
    <prince title="Andrew, Duke of York" />
    <!-- error: <prince title="Edward, Earl of Wessex" > -->
    <prince title="Edward, Earl of Wessex" />
</queen>
```

3a.



3b. 7.
title, marriedTo
"Charles, Prince of Wales","Lady Diana Spencer"

###3c) and 3d) are asking students to understand difference between elements and attributes and when to use one over the other. See e.g. https://www.ibm.com/developerworks/library/x-eleatt/

3c. Under this model, a person has only one title (structure is simple), can't have repeated titles with complex substructure.

3d. Can have a complex sub-structure, can have multiplicity/repeats (many princes)

4. Offers the opportunity to discuss about how namespaces work, what the default namespace does, how namespace prefixes are used, etc

```
<?xml version="1.0" encoding="utf-8"?>
<q:queen title="Queen Elizabeth II" marriedTo="Philip, Duke of
Edinburgh" xmlns="http://info.gov.uk"
xmlns:q="http://queenly.gov.uk"
xmlns:p="http://princely.gov.uk" xmlns:ps="princessly.gov.uk">
    <p:prince title="Charles, Prince of Wales" marriedTo="Lady
Diana Spencer">
        <p:prince title="Prince William of Wales" />
        <p:prince title="Prince Henry of Wales" />
    </p:prince>
    <ps:princess title="Anne, Princess Royal" />
    <p:prince title="Andrew, Duke of York" />
    <p:prince title="Edward, Earl of Wessex" />
</q:queen>
```

5a)

```
from lxml import etree # import the library

xmltree = etree.parse("royal.xml")
root = xmltree.getroot()

# Write a Python code to get the title property of queen's
grandsons.
for child in root: # iterate over prince and princess under
queen
    for grandson in child.iterchildren(tag="prince"):
        print (grandson.get('title'))

# Write a Python code to get the full title of the only
princess in the family tree.
the_only_princess = root.find("princess")
print (the_only_princess.get('title'))
```

5b)

```
xmltree = etree.parse("book.xml")
root = xmltree.getroot()
root.find("isbn").text='Unknown'
output = etree.tostring(root, pretty_print=True,encoding="UTF-
8")
open('book-new.xml','wb').write(output)
```

6.


```
{
    "id": "book001",
    "author": "Salinger, J. D.",
    "title": "The Catcher in the Rye",
    "price": "44.95",
    "language": "English",
    "publish_date": "1951-07-16",
    "publisher": "Little, Brown and Company",
    "isbn": "0-316-76953-3",
    "description": "A story about a few important days in the
life of Holden Caulfield"
}
```


7.


```
{
    "id": "book001",
    "author": "Salinger, J. D.",
    "title": "The Catcher in the Rye",
    "price": "44.95",
    "language": [
        "English",
        "Spanish",
        "German"
    ],
    "publish_date": "1951-07-16",
    "publisher": "Little, Brown and Company",
    "isbn": "0-316-76953-3",
    "description": "A story about a few important days in the
life of Holden Caulfield"
}
```



```
{
  "id": "book001",
  "author": "Salinger, J. D.",
  "title": "The Catcher in the Rye",
  "price": "44.95",
  "language": [
    "English",
    "Spanish",
    "German"
  ],
  "publish_date": [
```

```
    {
      "edition": "first",
      "date": "1951-07-16"
    },
    {
      "edition": "second",
      "date": "1979-01-01"
    }
  ],
  "publisher": "Little, Brown and Company",
  "isbn": "0-316-76953-3",
  "description": "A story about a few important days in the life of Holden
Caulfield"
}
```

8)

```
##this code is written to immediately follow the answer to week 2 workshop,
##for standalone use, may need to  import pandas and initialize countries

population = pd.read_csv('population.csv', index_col = 'Country', encoding = 'ISO-
8859-1')
cntryList  = ['Canada', 'United States', 'China', 'Australia']
# select the population information of the countries in cntryList between 1990 to
2013
cntryPop   = population.ix[cntryList,:-3]


# compute the average growth per year
sYear  = 1990
eYear  = 2013
period = eYear - sYear
# adding a new column to cntryPop
# the average growth is computed as (end-start)*100/(end*period)
cntryPop['growth'] = (population['2013'] p
population['1990'])*100/(population['2013']*period)


# select the emission information of the countries in cntryList between 1990 to
2013
emission  = pd.read_csv('emission.csv', index_col = 'Country', encoding = 'ISO-
8859-1')
cntryEmsn = pd.DataFrame((emission.ix[c,:-3] for c in cntryList))

cntryEmsn['SUM'] = cntryEmsn.ix[:, year[:8]].sum(axis=1)
cntryEmsn['Avg'] = cntryEmsn.ix[:, year[:8]].mean(axis=1)
cntryEmsn['growth'] = (cntryEmsn['2013'] -
cntryEmsn['1990'])*100/(cntryPop['1990']*period)
cntryEmsn
```

```
# selecting 2010 emission information
emission2010   = emission['2010']

# adding new columns to the countries
countries['emission']    = emission2010
grouped = countries.groupby('Region')
for k, group in grouped:
   print (k)
   # finding the top 10 emitting countries for each region
   top10 = group.sort_values(by = 'emission', ascending=False)[:10]
   # finding the Income group of the top 10 emitting countries for each region
   print (top10['IncomeGroup'].value_counts())


# Optional: more income means more co2 emission? or there may be other
explanations?
print ('-'*20)
population2010 = population['2010']
countries['population']  = population2010
countries['emission_per_capita'] = countries['emission']/countries['population']
grouped = countries.groupby('Region')
for k, group in grouped:
   print (k)
   top10 = group.sort_values(by = 'emission_per_capita', ascending=False)[:10]
   print (top10['IncomeGroup'].value_counts())
```