HW-6 - XML Queries In Python

CSCI 3287

Enter your name here:

Name: Nicolas Mavromatis

Identikey:nima6629@colorado.edu

collaborating with: Gurhar Khalsa

If you haven't already reviewed the LXML tutorial and played with the LXML notebook, you're encouraged to do so.

We're going to use to lxml library to access and manipulate XML data. Some of this material is an adaptation of the lxml tutorial.

The LXML library doesn't provide an XQuery interface, so we're going to use a tool called XQilla as a commandline interface.

In [58]:
```
import lxml
import lxml.etree as etree
import lxml.html
```

All of the problems (XPath, Xquery, XSLT) will use the **products** XML document below. I've created an LXML ElementTree for that document. I've also created a copy of the document in the file `/tmp/products.xml` that can be used for the XQuery exercise.

In [59]:
```
products_text = """<Products>
  <Maker name="A">
    <PC model="1001" price="2114">
      <Speed>2.66</Speed>
      <RAM>1024</RAM>
      <HardDisk>250</HardDisk>
    </PC>
    <PC model="1002" price="995">
      <Speed>2.10</Speed>
      <RAM>512</RAM>
      <HardDisk>250</HardDisk>
    </PC>
    <Laptop model="2004" price="1150">
      <Speed>2.00</Speed>
      <RAM>512</RAM>
      <HardDisk>60</HardDisk>
      <Screen>13.3</Screen>
    </Laptop>
    <Laptop model="2005" price="2500">
      <Speed>2.16</Speed>
      <RAM>1024</RAM>
      <HardDisk>120</HardDisk>
      <Screen>17.0</Screen>
    </Laptop>
  </Maker>
  <Maker name="E">
    <PC model="1011" price="959">
      <Speed>1.86</Speed>
      <RAM>2048</RAM>
      <HardDisk>160</HardDisk>
    </PC>
    <PC model="1012" price="649">
      <Speed>2.80</Speed>
      <RAM>1024</RAM>
      <HardDisk>160</HardDisk>
    </PC>
    <Laptop model="2001" price="3673">
      <Speed>2.00</Speed>
      <RAM>2048</RAM>
      <HardDisk>240</HardDisk>
      <Screen>20.1</Screen>
    </Laptop>
    <Printer model="3002" price="239">
      <Color>false</Color>
      <Type>laser</Type>
    </Printer>
  </Maker>
  <Maker name="H">
    <Printer model="3006" price="100">
      <Color>true</Color>
      <Type>ink-jet</Type>
    </Printer>
    <Printer model="3007" price="200">
      <Color>true</Color>
      <Type>laser</Type>
    </Printer>
  </Maker>
</Products>
"""
```

In [60]:

```
products = etree.XML(products_text)

with open("/tmp/products.xml","w") as f:
    f.write(products_text)
```

1) XPath [45 points total]

To do the XPath homework, you'll need to use the Product XML file.

1a) Find the amount of RAM on each PC [5 pts]

Your output should look like `['1024', '512', '2048', '1024']`

In [61]:

```
L=[]
for RAM in products.xpath('Maker/PC/RAM'):
    L.append(RAM.text)

print(L)
```

```
['1024', '512', '2048', '1024']
```

1b) Find the price of each product of any kind. [5 pts]

Your solution doesn't need to restrict itself to a PC or Printer -- it should report the price of *any* product.

In [62]:
```
L=[]
for p in products.xpath('Maker/*'):
    L.append(p.get('price'))

print(L)
```

```
['2114', '995', '1150', '2500', '959', '649', '3673', '239', '100', '200']
```

1c) Find the text of all the printer elements. [5 pts]

This should reduce the text of each element within a **Printer** element (e.g. 'false', 'laser', *etc*)

In [63]:
```
L=[]
for p in products.xpath('Maker/Printer'):
    L.append(p[0].text)
    L.append(p[1].text)

print(L)
```

```
['false', 'laser', 'true', 'ink-jet', 'true', 'laser']
```

1d) Find the makers of laser printers. [5 pts]

There are two makers of printers, 'E' and 'H'

In [64]:
```
#Find the boolean value of Type="laser", then back track to parent node and get its nam
L=[]
for m in products.xpath('*//Printer[Type="laser"]'):
    L.append(m.getparent().get('name'))

print(L)
```

```
['E', 'H']
```

1e) Find the makers of PC's and/or laptops. [5 pts]

There are two makers of PC and Laptops. Although your query could check for just one of PC or Laptop and be correct, you must construct your query to work even if one Maker only makes one or the other kind of product.

In [65]:
```
#Find the boolean value of Type="laser", then back track to parent node and get its nam
L=set()
for m in products.xpath('*//PC | *//Laptop'):
    L.add(m.getparent().get('name'))

print(L)
```

```
{'E', 'A'}
```

1f) Find the model numbers of PC’s with a hard disk of at least 200 gigabytes [10 pts]

In [66]:

```
L=[]
for h in products.xpath('*//HardDisk'):
    if(int(h.text)>=200):
        L.append(h.getparent().get('model'))

print(L)
```

```
['1001', '1002', '2001']
```

1g) Find the makers that make at least two PC's [10 pts]

This means that there are two PC's that are *siblings* in the element tree.

In [67]:

```
L=[]
for s in products.xpath('Maker[count(PC)>=2]'):
    L.append(s.get('name'))

print(L)
```

```
['A', 'E']
```

XQuery [45 Pts Total]

As mentioned, we need a completely different library and interface for XQuery and the Python Binding for that library is not very robust.

So, we're using to use the command line tool "xqilla" to run our queries.

Let's use an XQuery tutorial that appears at the W3Schools website. That tutorial uses a **books.xml** dataset. As before, we'll include here as a string so you can see dataset.

2a) Find the Printers models with a price less than 1000. [5 pts]

Your output should look like `['3002', '3006', '3007']`

In [68]:

```
### BEGIN SOLUTION
#Note: data() strips the element tags.
with open("/tmp/query.xq","w") as q:
    q.write('''\
let $x:= doc("/tmp/products.xml")/Products/Maker/Printer
for $p in $x
where ($p/@price<1000)
return data($p/@model)
''')
### END SOLUTION
%system xqilla /tmp/query.xq
```

Out[68]:

```
['3002', '3006', '3007']
```

2b) Find the Printer...[5 pts]

Find the Printer elements with a price <=200, and produce the sequence of the elements of these printers surrounded by a tag `<CheapPrinters>`

Your output should look like

```
['<CheapPrinters><Printer model="3006" price="100">',
 '      <Color>true</Color>',
 '      <Type>ink-jet</Type>',
 '    </Printer><Printer model="3007" price="200">',
 '      <Color>true</Color>',
 '      <Type>laser</Type>',
 '    </Printer></CheapPrinters>']
```

In [69]:

```
### BEGIN SOLUTION
with open("/tmp/query.xq","w") as q:
    q.write('''\
for $x in doc("/tmp/products.xml")/Products/Maker/Printer
where ($x/@price<=200)
return <CheapPrinters>{$x}</CheapPrinters>
''')
### END SOLUTION
%system xqilla /tmp/query.xq
```

Out[69]:

```
['<CheapPrinters><Printer model="3006" price="100">',
 '      <Color>true</Color>',
 '      <Type>ink-jet</Type>',
 '    </Printer></CheapPrinters>',
 '<CheapPrinters><Printer model="3007" price="200">',
 '      <Color>true</Color>',
 '      <Type>laser</Type>',
 '    </Printer></CheapPrinters>']
```

2c) Find the names of the makers of both printers and laptops. [5 pts]

You may want to read about quantification in XQuery

In [83]:

```
### This is a vague description, so I will assume this:
#IF you want a maker that SIMULTANEOUSLY makes both, include "and":
#ELSE include "or" (not done)
with open("/tmp/query.xq","w") as q:
    q.write('''\
for $x in doc("/tmp/products.xml")/Products/Maker
where ($x/Laptop and $x/Printer)
return data($x/@name)
''')
### END SOLUTION
%system xqilla /tmp/query.xq
```

Out[83]:

```
['E']
```

2d) Find the names of the makers that produce at least two PC's with a speed of 2.00 or more. [10 pts]

There's only one such maker (`A`). This problem exercises the Axes.

In [72]:

```
### BEGIN SOLUTION
with open("/tmp/query.xq","w") as q:
    q.write('''\
for $x in doc("/tmp/products.xml")/Products/Maker
where count(($x/PC[Speed>=2.00]))>=2
return data($x/@name)
''')
### END SOLUTION
%system xqilla /tmp/query.xq
```

Out[72]:

```
['A']
```

2e) Find the makers such that every PC they produce has a price no more than 1000. [10 pts]

Note that a maker that does not make PC's might statisfy a constraint that every PC make by that maker is less than $1000....

In [73]:

```
### HERE IS THE SOLUTION NOT INCLUDING H (Which does NOT produce PC's)
#-This makes more sense to me, but I do it BOTH WAYS
with open("/tmp/query.xq","w") as q:
    q.write('''\
for $x in doc("/tmp/products.xml")/Products/Maker
where (every $p in $x/PC/@price satisfies $p<=1000) and $x/PC
return data($x/@name)
''')
### END SOLUTION
%system xqilla /tmp/query.xq
```

Out[73]:

```
['E']
```

In [75]:

```
### HERE IS THE SOLUTION INCLUDING H (Which does NOT produce PC's)
#The difference is not checking "and $x/PC"
with open("/tmp/query.xq","w") as q:
    q.write('''\
for $x in doc("/tmp/products.xml")/Products/Maker
where (every $p in $x/PC/@price satisfies $p<=1000)
return data($x/@name)
''')
### END SOLUTION
%system xqilla /tmp/query.xq
```

Out[75]:

```
['E', 'H']
```

2f) Produce a sequence of elements of the form.. [10 pts]

`<Laptop> <Model>x</Model> <Maker>y</Maker> </Laptop>` where x is the model number and y is the name of the maker of the laptop.

Your output should look like

```
['<Laptop><Model>2004</Model><Maker>A</Maker></Laptop>',
 '<Laptop><Model>2005</Model><Maker>A</Maker></Laptop>',
 '<Laptop><Model>2001</Model><Maker>E</Maker></Laptop>']
```

In [76]:
```
### BEGIN SOLUTION
#Trick is to use for y in x, so no extra iterations are done
#If instead, I use x as full path length, it does double iterations.
with open("/tmp/query.xq","w") as q:
    q.write('''\
for $y in doc("/tmp/products.xml")/Products/Maker,
    $x in $y/Laptop/@model

return (<Laptop><Model>{data($x)}</Model><Maker>{data($y/@name)}</Maker></Laptop>)
''')
### END SOLUTION
%system xqilla /tmp/query.xq
```

Out[76]:
```
['<Laptop><Model>2004</Model><Maker>A</Maker></Laptop>',
 '<Laptop><Model>2005</Model><Maker>A</Maker></Laptop>',
 '<Laptop><Model>2001</Model><Maker>E</Maker></Laptop>']
```

XSLT [10 pts total]

You'll use LXML to perform the XSLT transformations. Most of the problems will ask you to present e.g. an HTML table. Rather than just look at the HTML, we'll use some Jupyter notebook tricks to enable us to display formatted HTML.

In [77]:
```
from IPython.core.display import display, HTML
```

In [78]:
```
table_data = '''<table>
                <tr> <th> A Table </th> <th> w/2 headers </th> </tr>
                <tr> <td> Some Data </td> <td> More Data </td> </tr>
                </table>'''
display(HTML(table_data))
```

A Table	w/2 headers
Some Data	More Data

3a) An HTML Table [5 pts]

The table should have a header “Manufacturers” followed by an enumerated list of the names of all the makers of products listed in the input.

In [79]:
```
### BEGIN SOLUTION
p3a_xslt = etree.XML('''\
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text"/>
  <xsl:template match = "/">
  <table>
      <tr> <th> Manufacturers </th> </tr>
     <xsl:for-each select = "/Products/Maker">
         <tr> <td> <xsl:value-of select= "@name"/> </td> </tr>
     </xsl:for-each>
  </table>
  </xsl:template>
</xsl:stylesheet>
''')
```

```
p3a_xform = etree.XSLT(p3a_xslt)
p3a_out=p3a_xform(products)

#print("Output is", etree.tostring(p3a_out, encoding=str))
display(HTML(etree.tostring(p3a_out, encoding=str)))
```

Manufacturers

A

E

H

3b) An HTML table [5 pts]

The table should include a row labeled Laptops, then print the model and price for each laptop. After that, it should print a row labeled PC and the model and price for each.

Your output should look like the following, but in HTML.

```
Laptops
Model   Price
2004    1150
2005    2500
2001    3673
PC
Model   Price
1001    2114
1002    995
1011    959
1012    649
```

In [80]:

```
### BEGIN SOLUTION
p3a_xslt = etree.XML('''\
<xsl:stylesheet xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text"/>
  <xsl:template match = "/">
  <table>
      <tr><th> Laptops </th></tr>
      <tr> <th> Model </th> <th> Price </th> </tr>
     <xsl:for-each select = "/Products/Maker/Laptop">
         <tr>
         <td> <xsl:value-of select= "@model"/> </td>
         <td> <xsl:value-of select= "@price"/> </td>
         </tr>
     </xsl:for-each>
     <PC/>
      <tr><th> PC </th></tr>
      <tr> <th> Model </th> <th> Price </th> </tr>
    <xsl:for-each select = "/Products/Maker/PC">
         <tr>
         <td> <xsl:value-of select= "@model"/> </td>
         <td> <xsl:value-of select= "@price"/> </td>
         </tr>
     </xsl:for-each>
  </table>
  </xsl:template>
</xsl:stylesheet>
''')
p3a_xform = etree.XSLT(p3a_xslt)
p3a_out=p3a_xform(products)

#print("Output is", etree.tostring(p3a_out, encoding=str))
display(HTML(etree.tostring(p3a_out, encoding=str)))
```

Laptops

Model	Price
2004	1150
2005	2500
2001	3673

PC

Model	Price
1001	2114
1002	995
1011	959
1012	649