CSC343 Worksheet 10 Solution

June 29, 2020

1. a) /Products/Maker/PC/RAM

   Notes:

   - XPATH and Selecting Nodes
     - nodename
       - Selects all nodes with the name "nodename"
     - /
       - Selects from the root node
     - //
       - Selects node in the document from the current node that match the selection no matter where they are
     - .
       - Select the current node
     - ..
       - Selects the parent of the current node
     - @
       - Selects attributes

     Example:

     ```
         /StarMovieData/Star//City

     ```

     - selects all City element in

       <StarMovieData>
         <Star>
           Here :)
         </Star>
       </StarMovieData>
   - Wildcards *

- Is used to say 'any tag'

Example:

```
/StarMovieData/*/@*

```

- '@*' means any attributes
- '*' means any tag

- Context of Expressions
  - [...] means that exists or there exists
  - [*integer*] selects ith child of its parent
  - [*Tag*] selects elements that have one or more sublements with 'Tag'
  - [*Attribute*] selects elements that have attribute 'Attribute'

  Example:

  ```
  /StarMovieData/Star[//City = "Malibu"]/Name

  ```

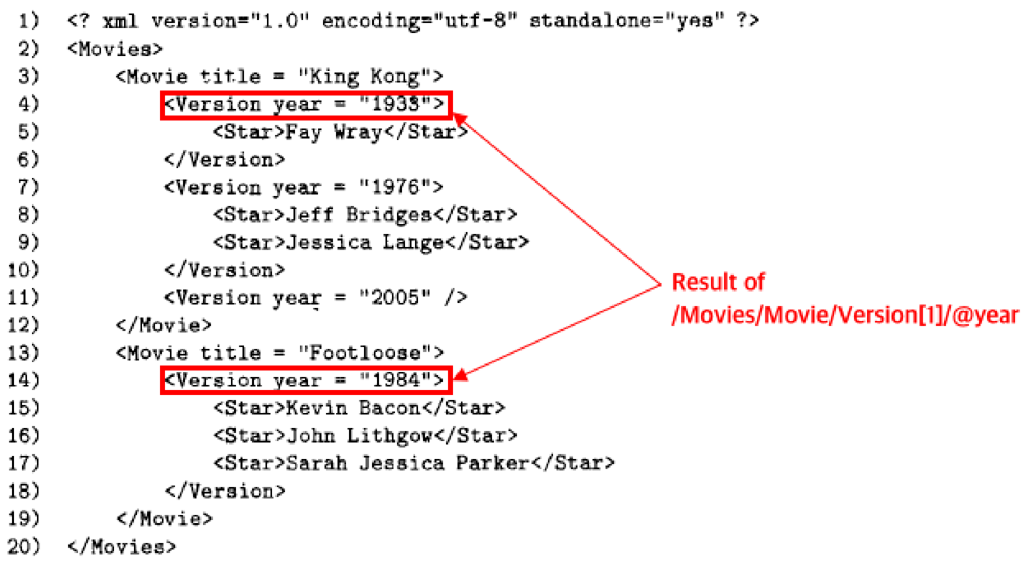
  * Means select all Star Name that contains City with value 'Malibu'

  Example:

  ```
  /Movies/Movie/Version[1]/@year

  ```

  * Returns value of 'year' attribute of first 'Version' tag in 'Movie'
  * e.g. 1933 and 1984



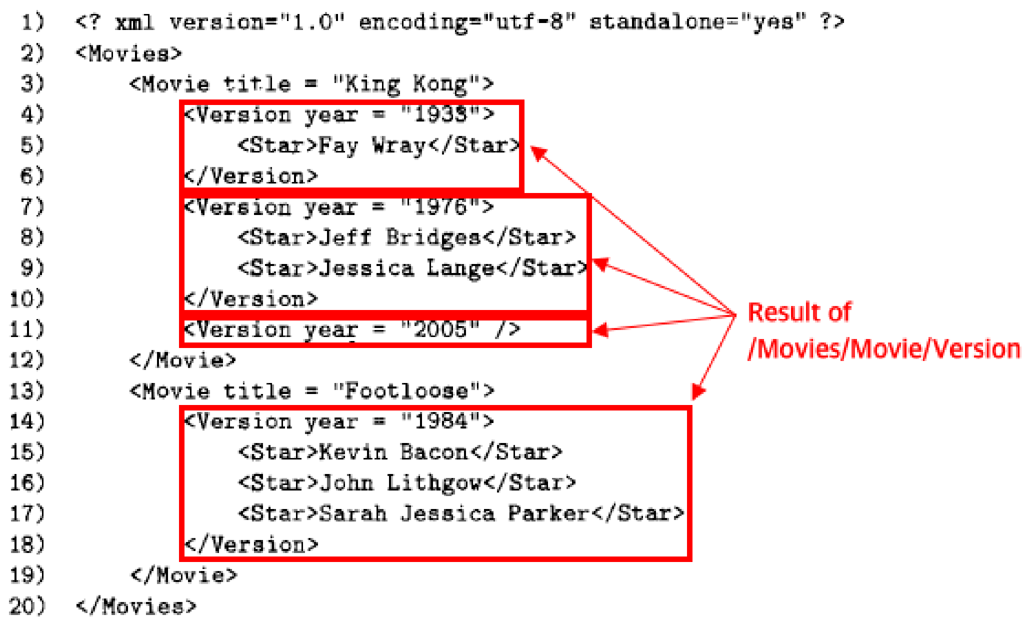
```
1)  <? xml version="1.0" encoding="utf-8" standalone="yes" ?>
2)  <Movies>
3)      <Movie title = "King Kong">
4)          <Version year = "1933">
5)              <Star>Fay Wray</Star>
6)          </Version>
7)          <Version year = "1976">
8)              <Star>Jeff Bridges</Star>
9)              <Star>Jessica Lange</Star>
10)         </Version>
11)         <Version year = "2005" />
12)     </Movie>
13)     <Movie title = "Footloose">
14)         <Version year = "1984">
15)             <Star>Kevin Bacon</Star>
16)             <Star>John Lithgow</Star>
17)             <Star>Sarah Jessica Parker</Star>
18)         </Version>
19)     </Movie>
20) </Movies>
```

Example 2:

```
    /Movies/Movie/Version

```

- ∗ Returns all ‘Version’ tag in ‘Movie’
- ∗ e.g. lines 4 through 6, 7 through 10, line 11, lines 14 through 18



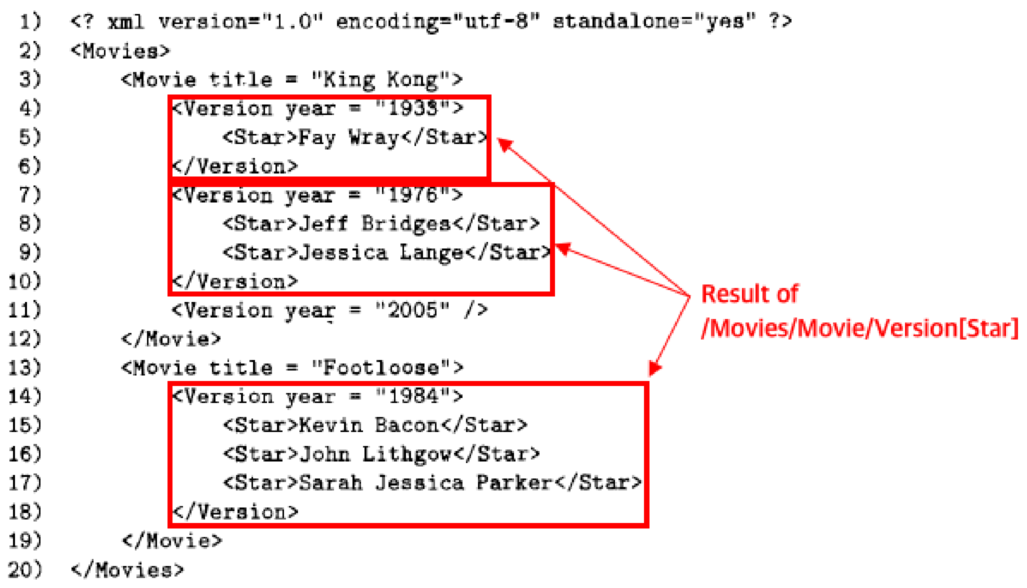
```
 1)  <? xml version="1.0" encoding="utf-8" standalone="yes" ?>
 2)  <Movies>
 3)      <Movie title = "King Kong">
 4)          <Version year = "1933">
 5)              <Star>Fay Wray</Star>
 6)          </Version>
 7)          <Version year = "1976">
 8)              <Star>Jeff Bridges</Star>
 9)              <Star>Jessica Lange</Star>
10)          </Version>
11)          <Version year = "2005" />
12)      </Movie>
13)      <Movie title = "Footloose">
14)          <Version year = "1984">
15)              <Star>Kevin Bacon</Star>
16)              <Star>John Lithgow</Star>
17)              <Star>Sarah Jessica Parker</Star>
18)          </Version>
19)      </Movie>
20)  </Movies>
```

Example 3:

```
    /Movies/Movie/Version[Star]

```

- ∗ Selects all ‘Version’ tag with one or more ‘Star’ tag inside
- ∗ e.g lines 4 through 6, 7 through 10, 14 through 18



```
<? xml version="1.0" encoding="utf-8" standalone="yes" ?>
<Movies>
    <Movie title = "King Kong">
        <Version year = "1933">
            <Star>Fay Wray</Star>
        </Version>
        <Version year = "1976">
            <Star>Jeff Bridges</Star>
            <Star>Jessica Lange</Star>
        </Version>
        <Version year = "2005" />
    </Movie>
    <Movie title = "Footloose">
        <Version year = "1984">
            <Star>Kevin Bacon</Star>
            <Star>John Lithgow</Star>
            <Star>Sarah Jessica Parker</Star>
        </Version>
    </Movie>
</Movies>
```

b) /Products/Maker/*/@price

c) /Products/Maker/Printer

d) /Products/Maker[/Printer/Type/text() = 'ink-jet']

Correct Solution:
/Products/Maker[Printer/Type/text() = 'ink-jet']

e) /Products/Maker[/PC | /Laptops]

Notes:

- XPATH and OR
  - **Syntax:** (*xpath expression 1*) | (*xpath expression 2*) [1]

References:

1) Stack Overflow, XPath OR operator for different nodes, link

f) /Products/Maker/*[HardDisk/text() > 200]/@model

2. a) /Ships/Class/Ship/@name

   b) /Ships/Class[@displacement > 35000]

   c) /Ships/Class/Ship[@launched < 1917]

   d) /Ships/Class/Ship[Battle/@outcome = 'sunk']/@name

   e) /Ships[Class/@name = Class/Ship/@name]/Class/Ship/@launched

   f) /Ships/Class/Ship[Battle]/@name

3. a)

```
    $products = doc("Products.xml");

    for $p in $products/Maker/Printer
        where @price < 100
        return $p

```

Notes:

- XQuery
  - Means **XML Query**
  - Is a functional language
- XQuery and FLWOR
  - FLWOR means
    1. **F**or - selects a sequence of nodes
    2. **L**et - binds a sequence to a variable
    3. **W**here - filters the nodes
    4. **O**rder By - sorts the nodes
    5. **R**eturn - what to return (gets evaluated once for every node)

    Example:

```
    doc("books.xml")/bookstore/book[price>30]/title

    for $x in doc("books.xml")/bookstore/book
    where $x/price>30
    return $x/title

```

  - **Let** cluase
    - **Syntax:** let *variable* := *expression*
    - Has a use case of storing document

      e.g.

      $stars := doc('stars.xml');
  - **For** cluase
    - **Syntax:** for *variable* in *expression*

    Example:

```
    let $movies := doc("movies.xml");
    for $m in $movies/Movies/Movie
    where $/@title = 'King Kong'
    return $m

```

- **Where** Clause
  - **Syntax: where** *condition*
- **Return** Clause
  - **Syntax: return** *expression*

- Replacement of Variables
  - Is done using curly braces {}

  Example:

```
let $movies := doc("movies.xml");
for $m in $movies/Movies/Movie
return <Movie title= {$/@title}>{$m/Version/Star}</Movie>

```

- Joins in XQuery
  - Is done using ‘,’ and where

  Example:

```
let $movies := doc("movies.xml"),
    $stars := doc("stars.xml")

for $s1 in $movies/Movies/Movie/Version/Star.
    $s2 in $stars/Stars/Star
where data($s1) = data($s2/Name)
return $s2/Address/City

```

- Elimination of duplicate values
  - Is done by enveloping query in function *distinct-values*

  Example:

```
let $movies := doc("movies.xml"),
$stars := doc("stars.xml")

let $starSeq := (for $s1 in $movies/Movies/Movie
                 return $m/Version/Star)
return <Star>{starSeq}</Star>

```

- Quantification in XQuery
  - **Syntax:** every *variable* in *expression1* satisfies *expression2*
    - Returns false if there is at least one item where expression1 makes expression2 false
  - **Syntax:** some *variable* in *expression1* satisfies *expression2*
    - Returns false if all items in expression1 makes expression2 false

Example

```
    let $stars := doc("stars.xml")
    for $s in $stars/Stars/Star
    where every $c in $s/Address/City satisfies
        $c = "Hollywood"
    return $s/Name

```

- Aggregations
  - can use *count*, *sum* or *max*

  Example:

```
    let $movies := doc("movies.xml")
    for $m in $movies/Movies/Movie
    where count($m/Version) > 1
    return $m


```

- Branching in XQuery Expressions
  - **Syntax:** if (*expression1*) then *expression2* else *expression3*

  Example:

```
    let $kk := doc("movies.xml")/Movies/Movie[@title = "King
  Kong"]
    for $v in $kk/Version
    return
        if ($v/@year = max($kk/Version/@year))
        then <Latest>{$v}</Latest>
        else <Old>{$v}</Old>

```

- Ordering the Result of a Query
  - **Syntax:** order *list of expressions*

  Example:

```
    let $movies := doc("movies.xml")
    for $m in $movies/Movies/Movie,
        $v in $m/Version
    order $v/@year
    return <Movie title = "{$m/@title}" year = "{$v/@year}" />

```

b)

```
    let $products := doc("products.xml")
    let $printerSeq := (for $p in $products/Maker/Printer
                        where $price < 100
                        return $p)
    return <CheapPrinters>{$p}</CheapPrinters>

```

c)
```
    let $products := /Products
    for $m in $products/Maker
        where exists($m/Printer) and exists($m/Laptop)
        return data($m/@name)
```

Correct Solution:

```
    let $products := doc("products.xml")
    for $m in $products/Maker
        where exists($m/Printer) and exists($m/Laptop)
        return data($m/@name)
```

d)
```
    let $products := doc("products.xml")
    for $m in $products/Maker
        where count($m/PC) >= 2 and $m/PC/Speed >= 3.00
        return data($m/@name)
```

e)
```
    let $products := doc("products.xml")
    for $m in $products/Maker
        where count($m/PC) >= 2 and $m/PC/Speed >= 3.00
        return data($m/@name)
```

f)
```
    let $products := doc("products.xml")
    for $m in $products/Maker
        where $m/PC/@price < 1000
        return $m
```

Correct Solution:

```
    let $products := doc("products.xml")
    for $m in $products/Maker
        where data($m/PC/@price) < 1000
        return $m
```

4. a)
```
    let $ships := doc("ships.xml")
    for $c in $ships/Class
        where data($c/@numGuns) > 10
        return $c
```

Correct Solution:

```
            let $ships := doc("ships.xml")
            for $c in $ships/Class
                where data($c/@numGuns) >= 10
                return $c
```

b)
```
    let $ships := doc("ships.xml")
    for $c in $ships/Class
        where data($c/@numGuns) > 10
        return data($c/Ship/@name)
```

Correct Solution:

```
            let $ships := doc("ships.xml")
            for $c in $ships/Class
                where data($c/@numGuns) >= 10
                return data($c/Ship/@name)
```

c)
```
    let $ships := doc("ships.xml")
    for $s in $ships/Class/Ship
        where data($s/Battle/@outcome) = 'sunk'
        return data($s/@name)
```

```
    let $ships := doc("ships.xml")
    for $c in $ships/Class
        where count($c/Ship) >= 3
        return data($c/@name)
```

d)
```
    let $ships := doc("ships.xml")
    for $c in $ships/Class
        where count($c/Ship) >= 3
        return data($c/@name)
```

f)
```
    let $ships := doc("ships.xml")
    for $c in $ships/Class
        where count($c/Ship/Battle) = 0
        return data($c/@name)
```

5.
```
    let $stars := docs("stars.xml")
    for $s in $stars/Star
    where exists($s/Address[./Street = "123 Maple St." and ./City = "
  Hollywood"])
    return $s/Name
```

6. This is impossible.

   We know that for some \$x in E to satisfy F to be false, there has to exist at least one item must be false.

   But since all items must be true for every \$x in E to satisfy F to be true, the two statement contradicts.