

# 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:

```
1 /StarMovieData/Star//City
2
```

- selects all City element in

```
<StarMovieData>
  <Star>
    Here :)
  </Star>
</StarMovieData>
```

- Wildcards \*

- Is used to say 'any tag'

### Example:

```
1 /StarMovieData/*/@*
2
```

- '@\*' 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:

```
1 /StarMovieData/Star[//City = "Malibu"]/Name
2
```

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

### Example:

```
1 /Movies/Movie/Version[1]/@year
2
```

- \* 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>

```

Result of  
/Movies/Movie/Version[1]/@year

**Example 2:**

```

1  /Movies/Movie/Version
2

```

- \* 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>

```

Result of  
/Movies/Movie/Version

**Example 3:**

```

1  /Movies/Movie/Version[Star]
2

```

- \* Selects all 'Version' tag with one or more 'Star' tag inside
- \* e.g. lines 4 through 6, 7 through 10, 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>

```

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

- 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:**  $(xpath\ expression\ 1) \mid (xpath\ expression\ 2)$  <sup>[1]</sup>

**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) 1  $products = doc("Products.xml");
      2
      3  for $p in $products/Maker/Printer
      4      where @price < 100
      5      return $p
      6

```

### 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**rders By - sorts the nodes
    5. **R**eturn - what to return (gets evaluated once for every node)

### Example:

```

1  doc("books.xml")/bookstore/book[price>30]/title
2
3  for $x in doc("books.xml")/bookstore/book
4      where $x/price>30
5      return $x/title
6

```

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

e.g.

```
$stars := doc('stars.xml');
```

- **For** clause
  - \* **Syntax:** *for variable in expression*

### Example:

```

1  let $movies := doc("movies.xml");
2  for $m in $movies/Movies/Movie
3      where $/@title = 'King Kong'
4      return $m
5

```

- **Where Clause**
  - \* **Syntax:** *where condition*
- **Return Clause**
  - \* **Syntax:** *return expression*
- Replacement of Variables
  - Is done using curly braces {}

**Example:**

```

1   let $movies := doc("movies.xml");
2   for $m in $movies/Movies/Movie
3   return <Movie title= {$/@title}>{$m/Version/Star}</Movie>
4

```

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

**Example:**

```

1   let $movies := doc("movies.xml"),
2   $stars := doc("stars.xml")
3
4   for $s1 in $movies/Movies/Movie/Version/Star,
5   $s2 in $stars/Stars/Star
6   where data($s1) = data($s2/Name)
7   return $s2/Address/City
8

```

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

**Example:**

```

1   let $movies := doc("movies.xml"),
2   $stars := doc("stars.xml")
3
4   let $starSeq := (for $s1 in $movies/Movies/Movie
5   return $m/Version/Star)
6   return <Star>{$starSeq}</Star>
7

```

- 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**

```

1  let $stars := doc("stars.xml")
2  for $s in $stars/Stars/Star
3  where every $c in $s/Address/City satisfies
4      $c = "Hollywood"
5  return $s/Name
6

```

- Aggregations

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

**Example:**

```

1  let $movies := doc("movies.xml")
2  for $m in $movies/Movies/Movie
3  where count($m/Version) > 1
4  return $m
5
6

```

- Branching in XQuery Expressions

- **Syntax:** if (*expression1*) then *expression2* else *expression3*

**Example:**

```

1  let $kk := doc("movies.xml")/Movies/Movie[@title = "King
    Kong"]
2  for $v in $kk/Version
3  return
4      if ($v/@year = max($kk/Version/@year))
5      then <Latest>{$v}</Latest>
6      else <Old>{$v}</Old>
7

```

- Ordering the Result of a Query

- **Syntax:** order *list of expressions*

**Example:**

```

1  let $movies := doc("movies.xml")
2  for $m in $movies/Movies/Movie,
3      $v in $m/Version
4  order $v/@year
5  return <Movie title = "{$m/@title}" year = "{$v/@year}" />
6

```

```

b) let $products := doc("products.xml")
2  let $printerSeq := (for $p in $products/Maker/Printer
3                      where $price < 100
4                      return $p)
5  return <CheapPrinters>{$p}</CheapPrinters>
6

```