XTraining

Part 2: XPath Operators

Regine Heberlein 5/18/2018

Homework Discussion

Write an XPath to find:

1. All unittitle elements whose value equals “Malice in Wonderland” (hint: consider child elements and use predicate [.=“Malice in Wonderland”])
2. Any elements within a c whose value equals “Tartuffe” (hint: use a wildcard)
3. All emph elements whose value equals “Tartuffe” and that are descendants of the 36th c element.

Use Outline View to double-check your results!

Homework Discussion

1. Interpreting the question. What do we mean by “value”?

→ Strict interpretation:

//unittitle[.=‘Malice in Wonderland’] ← empty sequence

//unittitle/*[.='Malice in Wonderland']

//unittitle/child::*[.='Malice in Wonderland']

→how about returning the value of unittitle whose child elements equal malice in wonderland

//unittitle[*='Malice in Wonderland']

2. //c//*[.='Tartuffe'] ← note the double slash

3. //c[36]//emph[.='Tartuffe'] ← predicate at every step

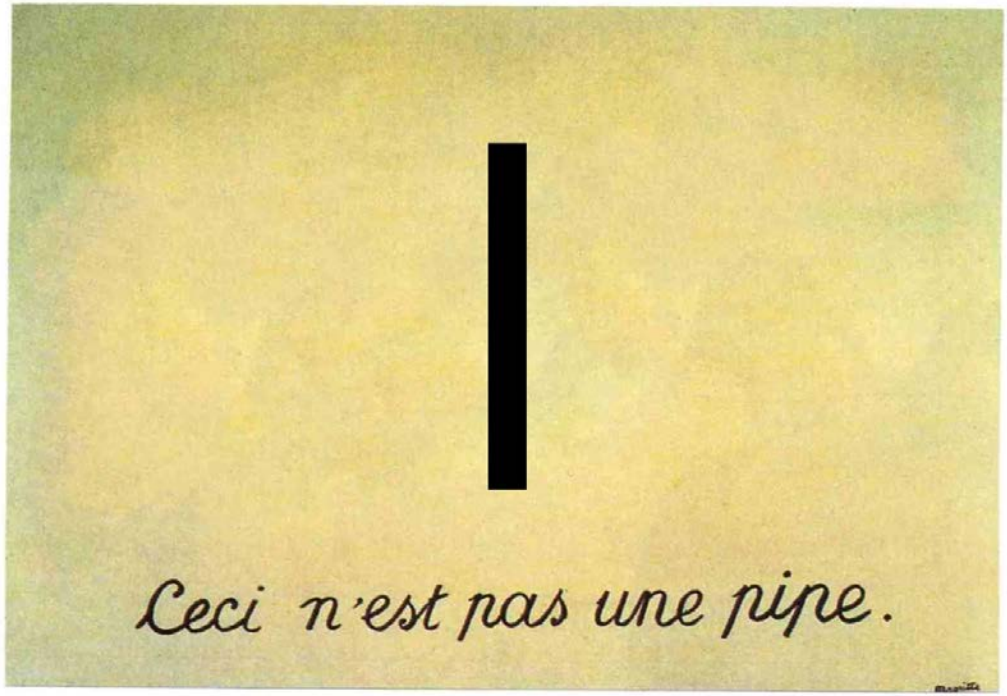
Expression Operators: Recap

@	attribute axis	publicationstmt/@id
*	element wildcard	address/*
@*	attribute wildcard	publicationstmt/@*
/	child axis (one step)	address/addressline
//	descendant axis	publicationstmt//*
.	self axis	address/./addressline
..	parent axis	address/../*
[]	predicate	language[@langcode]

Watch your step!

//publicationstmt/@id
//publicationstmt/@*
//address/*
//address/addressline
//publicationstmt//*
//address/./addressline
//address/../*
//address/..//*
//language[@langcode=“eng”]

I (Union)



How Union Works

```
//eadid/text() | //archdesc/did/unitid/text()
//unitdate | //unittitle
//repository[@id ] | //unittitle[@encodinganalog]
//repository[@id  | @encodinganalog]
//repository[@id  | @encodinganalog | @foo]
```

→Return the union of all extents and containers

```
//extent | //container
```

$ (Variable)

`for` expressions and quantified expressions

```
for $x in //did return $x
for $x in //address return $x/addressline
for $x in //dsc return $x//unitdate
for $x in //ead return $x//persname[@role="cre"]
for $x in //unitdate[@normal] return $x/@*[name(.)!='normal']
```

Quantified Expressions

every $x in //language satisfies $x/@lang

some $x in //language satisfies $x/@altrender

→ wait… what?

some $x in //emph satisfies $x/parent::unittitle

every $x in //emph satisfies $x/parent::unittitle

→ wait… what?

Where is your boat?

some $x in unitdate satisfies $x/@normal v.

some $x in //unitdate satisfies $x/@normal v.

some $x in //unitdate satisfies @normal

some $x in //unitdate satisfies not($x/@normal)

|| (Concatenation) – New in XPath 3.0

//did/(unittitle || unitdate)
//did/(unittitle || ' ' || unitdate)

Using the concatenation operator and variable operator, return the unittitle for each series along with a count of direct child components (use function `count($x/c)`).

Format your result for ease of reading!

for $x in //c[@level='series'] return $x/did/unittitle || ' size: ' || count($x/c) || ' child components'

! (Mapping) – New in XPath 3.0

//unittitle/string()/normalize-space(.)

… no can do

//unittitle ! string() ! normalize-space()

How Mapping Works

Using the mapping operator, get this to work:

//unittitle/string()/lower-case(.)

→ //unittitle/string() ! lower-case(.)

=> (Function Chaining) – New in XPath 3.1 (not implemented in oXygen 19)

```
contains(
        normalize-space(
                substring-after(
                        //titleproper, ':')
                                ),
                                        'Aid')

//titleproper
        => substring-after(':')
        => normalize-space()
        => contains('Aid')
```

Boolean Operators

or	Tests whether either the first or second expressions are true. If the first expression is true, the second is not evaluated.
and	Tests whether both the first and second expressions are true. If the first expression is false, the second is not evaluated.
=	Tests whether two expressions are equal.
!=	Tests whether the two expressions are not equal.
<	Tests whether the first expression is less than the second.
>	Tests whether the first expression is greater than the second.
<=	Tests whether the first expression is less than or equal to the second.
>=	Tests whether the first expression is greater than or equal to the second.

Caveat: union vs. or

//controlaccess/* or //unittitle

//unitdate or //foo

//foo or //bar

//repository[@id] or //unittitle[@encodinganalog]

//repository[@id] | //unittitle[@encodinganalog]

//repository[@id | @altrender]

//repository[@id or @altrender]

Exercise

//unittitle and //unitdate

//controlaccess/* or //unittitle

//unitdate[@normal and @*[name(.)!='normal']]

//c[@level='file']

//c/@level[.!='file']

count((//c[@level='series'])[1]/c) <=
count((//c[@level='series'])[2]/c)

count((//c[@level='series'])[1]/c) >=
count((//c[@level='series'])[2]/c)

Mathematical Operators

+	*plus*	Adds one number to another	2+3
-	*minus*	Subtracts one number from another	5-2
*	*multiplication*	Multiplies one number by another	2*3
div	*division*	Performs a floating-point division between two numbers	6/2
mod	*modulo*	Returns the floating-point remainder of dividing one number by another	6 mod 4

Exercise

How many components does each series have on average?

→hint: use count() to get the component count

count(//c[@level!='series']) div
count(//c[@level='series'])

String Alert

simple query fails:

```
//extent>1
```

see why:

```
//extent/number(.)
```

take a look:

```
//extent
```

parse it out:

```
//extent/replace(., '\p{L}', '')
```

Challenge

The Dread Pirate Roberts has hidden treasure on the island of AC387. Find it and tally the number of pearls it contains as well as how much they are worth!