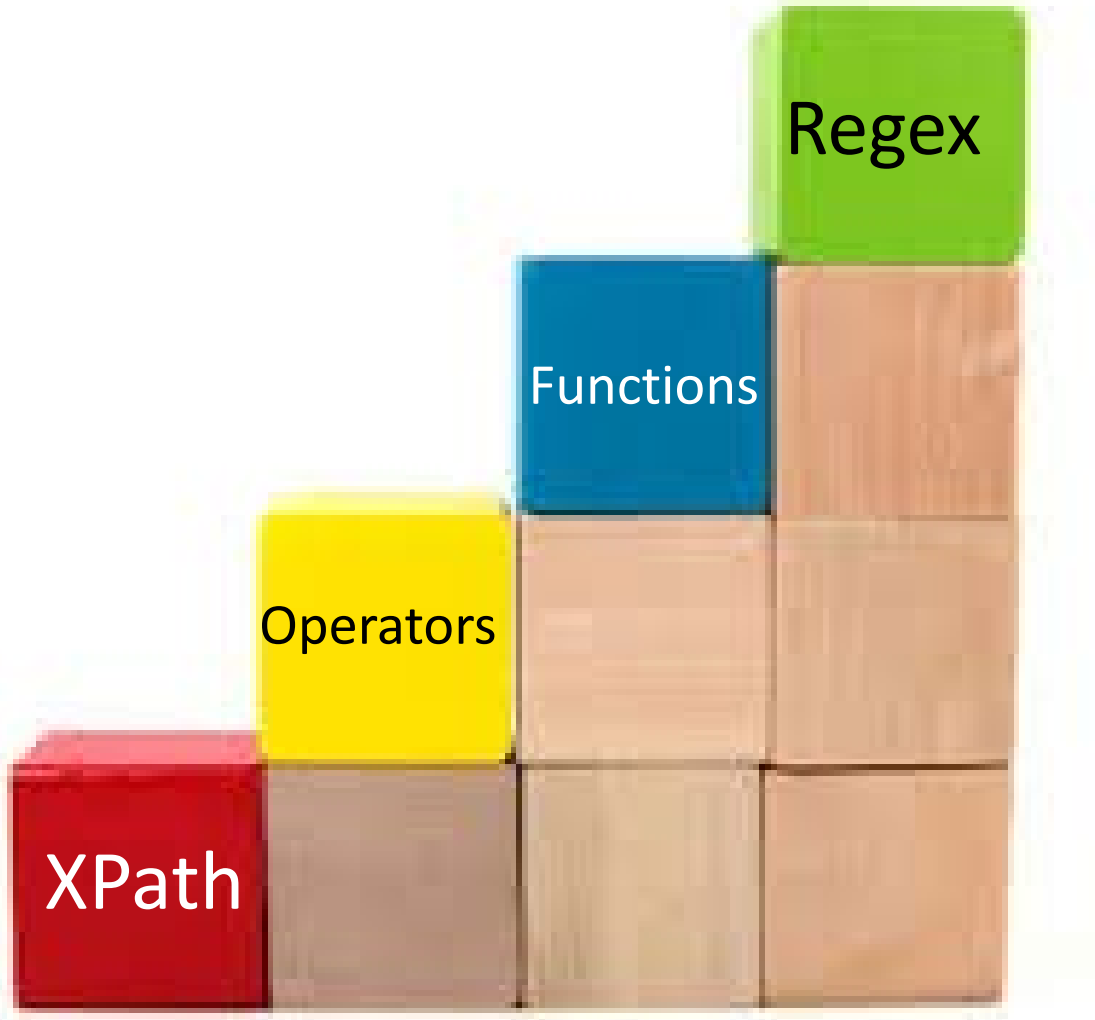
Before ~~Peter~~ xQuery



Easing into XPath

- Node-set: //unittitle
- String: //unittitle/string()
- Boolean: //unittitle or //unitdate
- Number: //unittitle/position()

Axis::node_test[predicate]

- Predicate may by anywhere in the location path

  //c[@level=“series”]/did/unittitle

  //c[@level=“series”]/did/unitdate[not(@normal)]

- One step may have multiple predicates

  //container[ancestor::c[1][@level="file"]][@type='box']

CAVEAT:
The order of predicates matters!

//c[1][@level="file"]

→ “return the first c of the context node IF its level attribute is set to file”

//c[@level="file"][1]

→ “return the first c of the context node whose attribute level is set to file”

CAVEAT: First of what?

//c[1]

(//c)[1]

Expression Operators

//publicationstmt/@id

//address/*

//publicationstmt/@*

//address/./addressline

//address/../*

//language[@langcode=“eng”]

What does this return?

//controlaccess//*[@encodinganalog > 650]

Expression Operator: $ (Variable)

- XPath 2.0:
  `for` expressions and quantified expressions

```
for $x in //did return $x
for $x in //address return $x/addressline
for $x in //ead return $x//persname[@role="cre"]
for $x in //unitdate[@normal] return $x/@*[not(name(.)="normal")]
```

Quantified Expressions: True or False?

every $x in //controlaccess satisfies $x/persname

some $x in //controlaccess satisfies $x/persname

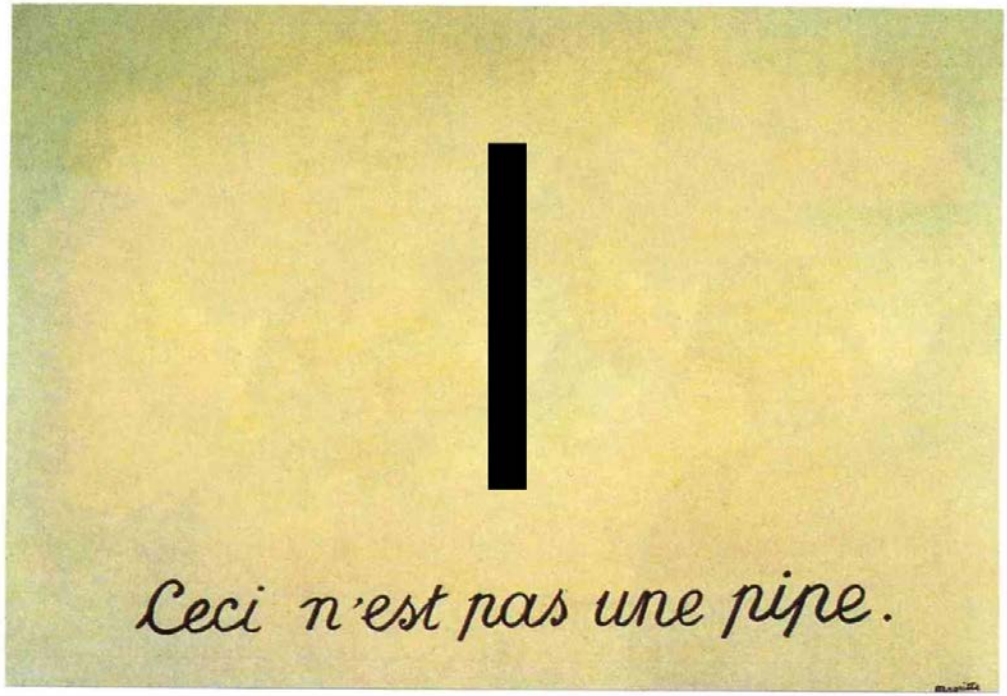
CAVEAT: Behavior of Quantified Expressions Executed on Empty Sequences

every $x in //controlacccess satisfies $x/persname

→ *true*

"Note that if the input sequence is empty, the "some" expression will always be false, while the "every" expression will always be true. This may not be intuitive to everyone, but it is logical—the "every" expression is true if there are no counter-examples; for example, it's true that every unicorn has one horn, because there are no unicorns that don't have one horn. Equally, and this is where the surprise comes, it is also true that every unicorn has two horns." –Michael Kay, XSLT 2.0 and XPath 2.0, 4th ed. (2008)

Expression Operator: | (Union)



How Union Works

```
//eadid/text() | //archdesc/did/unitid/text()
```

```
for $x in //ead return $x//eadid/text() |
        $x/archdesc/did/unitid/text()
```

Boolean Operators

or	//controlaccess/* or //unittitle
and	//unittitle and //unitdate
=	//c[@level="file"]
!=	//unitdate[@normal and @*[name(.)!="normal"]]
<	
<=	count((//c[@level="series"])[1]/c) <= count((//c[@level="series"])[2]/c)
>	
>=	count((//c[@level="series"])[1]/c) >= count((//c[@level="series"])[2]/c)

CAVEAT: union vs. or

//repository/@id or //repository/@altrender

//repository/@id | //repository/@altrender

What happens if you put “|” in a predicate?

//repository[@id | @altrender]

Mathematical Operators

+

-

*

div

mod

On avg, how many files per series?

count(//c[@level="file"]) div count(//c[@level="series"])

Functions!!!

not() //repository[not(address)]

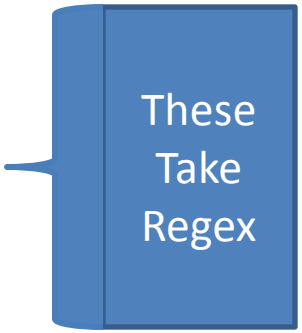
count() count(//unitdate[not(@normal)])

contains() count(//unitdate[ancestor::dsc and contains(., "undated")])

matches()

tokenize()

replace()



More Functions!

```
//did//text()[.=normalize-space()]

//c[substring-after(@id, "C1468_c")[number(.)=500]]

count(//c[substring-after(@id, "C1468_c")[number(.)>500]])

max(//unittitle/string-length())

//unitdate/replace(normalize-space(.), '^undated$', 'has  anyone bothered to check these unitdates??')
```

Yet More Functions!

- http://www.w3schools.com/xpath/xpath_functions.asp

- http://www.xqueryfunctions.com/

CAVEAT: Empty Sequence with function not() v. boolean !=

//dsc/c[1]/c[1]/c[5]/did[1]/container[3]!=6

Returns false (container[3] doesn't exist)

not(//dsc/c[1]/c[1]/c[5]/did[1]/container[3]=6)

Returns true(non-existent container[3] is not 6)

Regex

http://www.regular-expressions.info/

http://regexpal.com/

http://regexbuddy.com/ ($$)

Regex Metacharacters

.	any character
\	escape character
|	or
?	zero or one
*	zero or more
+	one or more

()	group
[]	range
[^]	negative range
{}	count
^	anchor: start of string
$	anchor: end of string

Escaping Sequences

\d	any digit
\D	any non-digit
\w	any word character
\W	any non-word char.
\s	any whitespace char.
\S	any non-whitespace char.
\t	tab
\n	newline
\r	carriage return
\p{}	match characters in category
\P{}	match characters not in category

Greedy Quantifiers (Add “?” To Make Them Lazy)

+

*

?

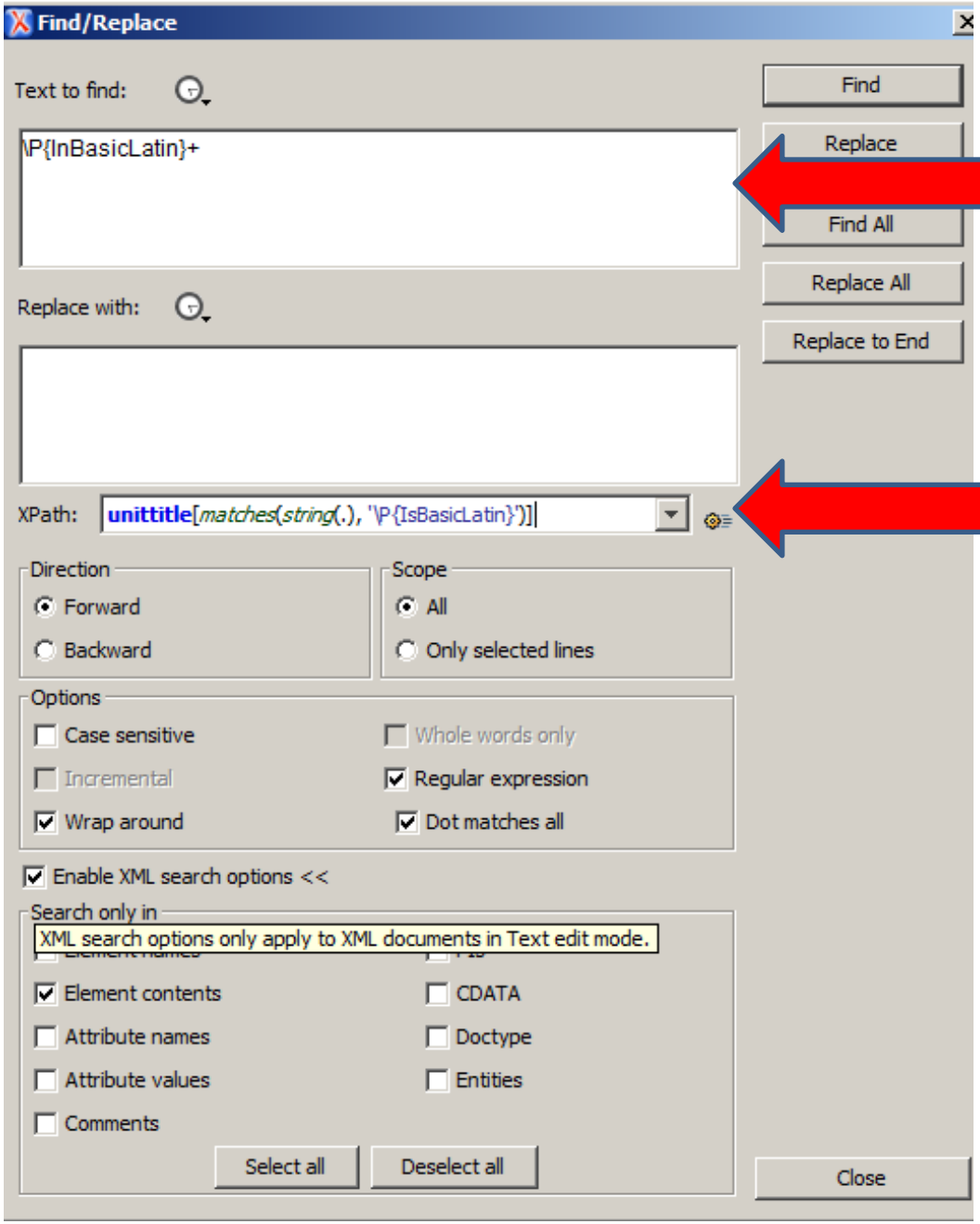
{2,}

CAVEAT

.+

v.

[\D\S]+



NB: Search box takes Java regex

NB: Xpath dialog takes XML regex

CAVEAT

Charl[^\W]{3}[\D\S]+

> “any character that is either not a digit or not whitespace [i.e., anything]”

Charl[^\W]{3}[^\d\s]+

> “any character that is neither a digit nor whitespace”

Charl[^\W]{3}[^\D\S]+

> “any character that is neither not a digit nor not whitespace [i.e., nothing]”

Regular Expression Flags

i	case insensitive
s	dot matches all (including \n)
m	multiline mode
x	ignore whitespace

```
//unittitle[matches(., ‘bern.+?\s', 'is')]
```

What does this find?

Use Find/Replace

Find: \p{L}+

XPath:

unitdate[not(matches(text(), 'undated', 'im'))]/text()

1. Find any unittitles containing characters not in the Basic Latin Block.
2. Find any characters in unittitle that are not in the Basic Latin Block.
3. Find any characters in unittitle that are in Latin-1 Supplement
4. Find any characters in unittitle that are neither in Basic Latin nor in Latin-1 Supplement
5. Find any unittitles containing characters from both Latin-1 Supplement and General Punctuation

1. [\D\S]+

unittitle[matches(string(.), '\P{IsBasicLatin}')] → XPath filter uses XML regex engine

2. \P{InBasicLatin}+ →Find/Replace uses Java regex engine

unittitle

3. \p{InLatin-1Supplement}+

unittitle

4. [^\p{InBasicLatin}\p{InLatin-1Supplement}]+

unittitle

5. [\D\S]+

unittitle[matches(string(.), '\p{IsLatin-1Supplement}') and matches(., '\p{IsGeneralPunctuation}')]

Replacing with named groups

Group using ()

Find: (Charl)(otte)

Replace: $1es

Play with matches()!

1. Find any unittitle that contains any number
2. Find any unittitle that contains a year

   (can you make it between 1600-2099?)
3. Find any unittitle that contains a year range

1. //unittitle[matches(., '\d+')]

2. //unittitle[matches(., '\d{4}')]

   or

   //unittitle[matches(., '(1[6-9]|20)\d{2}')]

3. //unittitle[matches(., '(1[6-9]|20)\d{2}-(1[6-9]|20)\d{2}')]

tokenize()

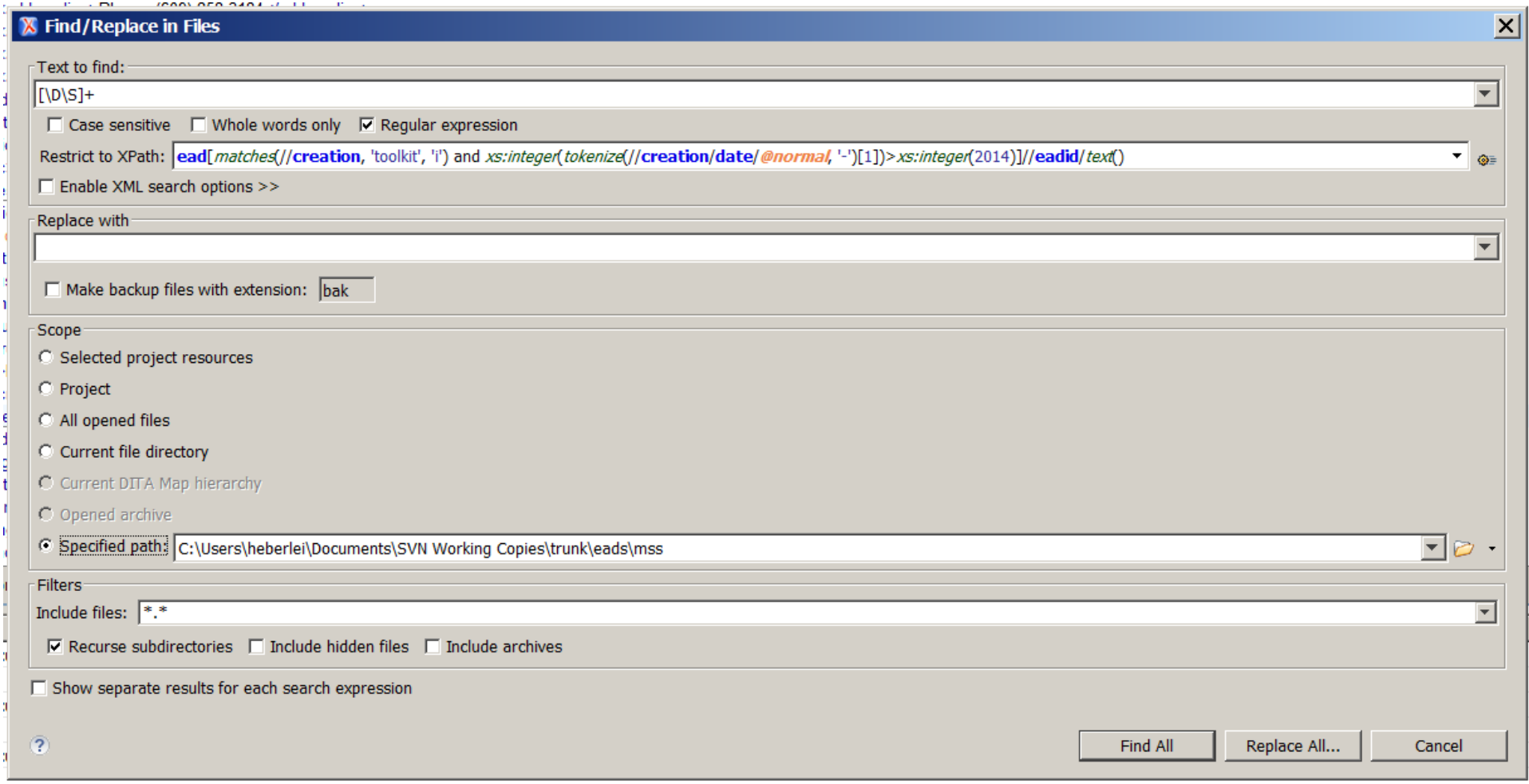
```
//unitdate[matches(., '^\d{4}-\d{4}$')]/tokenize(., '-')
```

Capstone Project!

From your Find/Replace in Files dialog, find any EAD's in a given repository that were created using Archivists' Toolkit after 2014.

Hint: try a simple approach using integers.

Help is on the next slide…


Find/Replace in Files
Text to find:
[\D\S]+
Case sensitive
Whole words only
Regular expression
Restrict to XPath:
ead[matches(//creation, 'toolkit', 'i') and xs:integer(tokenize(//creation/date/@normal, '-')[1])>xs:integer(2014)]//eadid/text()
Enable XML search options >>
Replace with
Make backup files with extension:
bak
Scope
Selected project resources
Project
All opened files
Current file directory
Current DITA Map hierarchy
Opened archive
Specified path:
C:\Users\heberlei\Documents\SVN Working Copies\trunk\eads\mss
Filters
Include files:
.
Recurse subdirectories
Include hidden files
Include archives
Show separate results for each search expression
Find All
Replace All...
Cancel