Exselt

a distributed streaming xslt 3.0 processor

Efficient XML processing with higher order functions

xmlprague

Abel Braaksma, Abrasoft consulting

Outline

- **Introducing higher order functions**
- Function items, partial function application and closures
- Efficiency techniques with XSLT 3.0
- Summary

Definitions

- First class functions – first class citizens

  "A programming language is said to have first-class functions if it treats functions as first-class citizens"

  Christopher Strachey, 1964

- What is a higher order function (HOF)?

  "A higher order function is a function that accepts functions as arguments, and/or that returns a function as a result."

  Gerard O'Regan, A Brief history of Computing

Definitions

- First class functions – first class citizens

  "A programming language is said to have first-class functions if it treats functions as first-class citizens"

  Christopher Strachey, 1964

- What is a higher order function (HOF)?

  "A higher order function is a function that accepts functions as arguments, and/or that returns a function as a result."

  Gerard O'Regan, A Brief history of Computing

Analogy: signing a paper

Second-class citizen

```
let $signatory:= "Schönfinkel"
return
    for $paper in $papers
    return f:SignPaper($paper, $signatory)
```

Imagine that

- The signatory shows up in person
- With his tools like pen and ink

First-class citizen:

```
<xsl:function name="f:getSignatory">
    <xsl:sequence select="
    function($paper) {
        (: implementation :) } " />
</xsl:function>
```

```
for $paper in $papers
return f:getSignatory()($paper)
```

First-class citizen:

```
<xsl:function name="f:getSignatory">
    <xsl:sequence select="
    function($paper) {
        (: implementation :) } " />
</xsl:function>
```

```
for $paper in $papers
return f:getSignatory()($paper)
```


(PL# 109-282, 120 Stat. 1186).
Signature
Date

Outline

- Introducing higher order functions
- **Function items, partial function application and closures**
- Efficiency techniques in XSLT 3.0 + HOF
- Summary

Function items

```
let $fun := function($a) { $a * $a - 1 }
return $fun(23)
```

Function items

```
let $fun := function($a) { $a * $a – 1 }
return $fun(23)

let $fun := function($a as xs:integer)
            as xs:integer
            {
                $a * $a – 1
            }
```

Function items

```
let $fun := function($a) { $a * $a – 1 }
return $fun(23)

let $fun := function($a as xs:integer)
            as xs:integer
            {
                $a * $a – 1
            }

<xsl:variable name="fun" as="function(*)"
    select="function($a) { $a * $a – 1 }" />

<xsl:sequence select="$fun(23)" />
```

First class citizens indeed

```
<xsl:apply-templates select="$fun" />
```

First class citizens indeed

```
<xsl:apply-templates select="$fun" />

<xsl:template match="~function(*)">
   <xsl:value-of select=".(23)" />
</xsl:template>
```

First class citizens indeed

```
<xsl:apply-templates select="$fun" />

<xsl:template match="~function(*)">
   <xsl:value-of select=".(23)" />
</xsl:template>

<xsl:template
   match="~function(*)[function-arity(.) lt 2]">
   <xsl:value-of select="
      if(function-arity(.) = 0) then .()
      else .(23)" />
</xsl:template>
```

First class citizens indeed?

- No easy type introspection
- No serialization
- No identity

Currying

Haskell Curry in his paper in 1930

Schönfinkeling

But: Schönfinkel pioneered this in his talk in 1920

Partial Function Application

in XPath 3.0, 2013

Partial Function Application

```
<xsl:variable name="quote"
    select="concat('&quot;', ?, '&quot')" />
<xsl:value-of
    select="movie/$quote(@title)" />

Result: ('"Harold and Maud"', '"9"', '"Milk"')
```

Partial Function Application

Closures

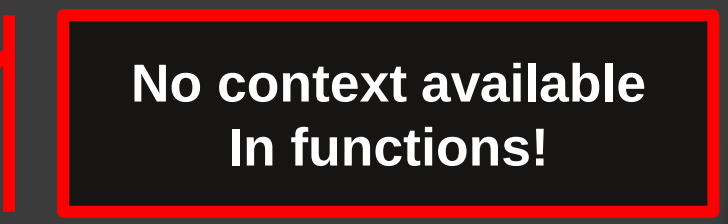
Inline functions remember their environment

Closures

```
<xsl:variable name="isbn-functions"
    select="
      book/(
        function()
        {
           @isbn
        } )" />
```

Closures

```
<xsl:variable name="isbn-functions"
    select="
      book/(
        function()
        {
           @isbn
        } )" />
```



Closures

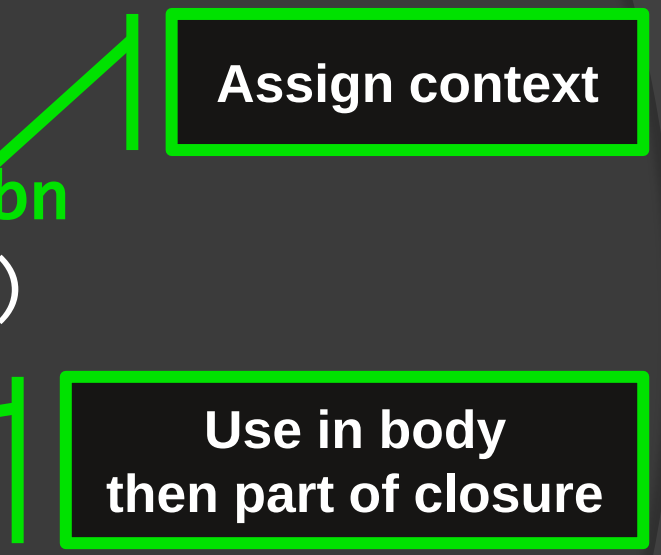
```
<xsl:variable name="isbn-functions"
    select="
      book/(
        let $isbn := @isbn
        return function()
        {
           $isbn
        } ) "/>

<xsl:apply-templates
     select="$isbn-functions" />
```

Closures

```
<xsl:variable name="isbn-functions"
    select="
      book/(
        let $isbn := @isbn
        return function()
        {
           $isbn
        } ) "/>
```

Closures

```
<xsl:variable name="isbn-functions"
    select="
      book/(
        let $isbn := @isbn
        return function()
        {
           $isbn
        } ) "/>
```



Closures

```
<xsl:variable name="isbn-functions"
    select="
      book/(
        let $isbn := @isbn
        return function()
        {
           $isbn
        } ) "/>

<xsl:apply-templates
     select="$isbn-functions" />
```

Closures

```
<xsl:template match="~function()">
    <xsl:value-of
        select=".()" />
</xsl:template>
```

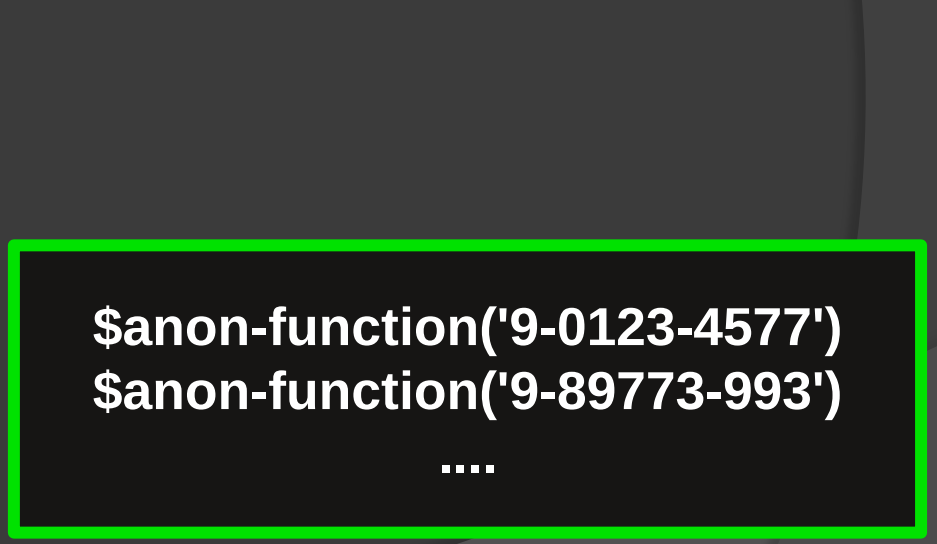
Closures

```
<xsl:template match="~function()">
   <xsl:value-of
      select=".()" />
</xsl:template>
```

matches only parameterless functions

Closures

```
<xsl:template match="~function()">
    <xsl:value-of
        select=".()" />
</xsl:template>
```



Outline

- Introducing higher order functions
- Function items, partial function application and closures
- **Efficiency techniques in XSLT 3.0 + HOF**
- Summary

Memoization

- Exchanges time for space
- cache = " full | partial | none "
- identity-sensitive = " yes | no "

Memoization

```
<xsl:function cache="full" name="f:factorial">
    <xsl:param name="x" />
    <xsl:sequence select="
          if($x = 0) then 1
          else $x * f:factorial($x – 1) " />
</xsl:function>
```

Memoization

```
<xsl:function cache="full" name="f:factorial">
   <xsl:param name="x" />
   <xsl:sequence select="
        if($x = 0) then 1
        else $x * f:factorial($x – 1) " />
</xsl:function>

let $format := concat(?, '! = ', ?, '&#A;')
return for $n in 0 to 100
       return $format($n, f:factorial($n))
```

Memoization

1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800


What now?
Stack overflow!
All rights reserved - rustedreality.com
by Stian Kruger ©

Advanced: continuation

- Continuation passing style, or CPS
- Trade stack space for heap space
- Make functions TCO that are multiple recursive
- Difficult to read and master

Advanced: continuation

```
<xsl:function name="f:fib">
    <xsl:param name="n" as="xs:integer">

    <xsl:sequence select="
        if ($n = 1 or $n = 2)
        then 1
        else
            f:fib($n - 1)
            + f:fib($n - 2)" />
</xsl:function>
```

Advanced: continuation

```
<xsl:function name="f:fib">
    <xsl:param name="n" as="xs:integer" />
    <xsl:param name="acc" as="function(*)" />

    <xsl:sequence select="
        if ($n = 1 or $n = 2)
        then $acc(1)
        else
            f:fib($n - 1, function($x) {
                f:fib ($n - 2, function($y) {
                    $acc($x + $y)
                })
            })" />
</xsl:function>
```

Advanced: continuation

```
<xsl:function name="f:fib">
    <xsl:param name="n" as="xs:integer" />
    <xsl:param name="acc" as="function(*)" />

    <xsl:sequence select="
        if ($n = 1 or $n = 2)
        then $acc($n)
        else
            f:fib($n - 1, function($x) {
                f:fib ($n - 2, function($y) {
                    $acc($x + $y)
                })
            })" />
</xsl:function>
```

Advanced: continuation

```
<xsl:function name="f:fib">
    <xsl:param name="n" as="xs:integer" />
    <xsl:param name="acc" as="function(*)" />

    <xsl:sequence select="
        if ($n = 1 or $n = 2)
        then $acc($n)
        else
            f:fib($n - 1, function($x) {
                f:fib ($n - 2, function($y) {
                    $acc($x + $y) }
                )}
            )" />
</xsl:function>
```

Advanced: continuation

```
<xsl:function name="f:fib">
    <xsl:param name="n" as="xs:integer" />
    <xsl:param name="acc" as="function(*)" />

    <xsl:sequence select="
        if ($n = 1 or $n = 2)
        then $acc($n)
        else
            f:fib($n - 1, function($x) {
                f:fib ($n - 2, function($y) {
                    $acc($x + $y)
                })
            })" />
</xsl:function>
```

Advanced: continuation

```
<xsl:function name="f:fib">
    <xsl:param name="n" as="xs:integer" />
    <xsl:param name="acc" as="function(*)" />

    <xsl:sequence select="
        if ($n = 1 or $n = 2)
        then $acc($n)
        else
            f:fib($n - 1, function($x) {
                f:fib ($n - 2, function($y) {
                    $acc($x + $y)
                })
            })" />
</xsl:function>
```

Advanced: continuation

```
<xsl:function name="f:fib">
    <xsl:param name="n" as="xs:integer" />
    <xsl:param name="acc" as="function(*)" />

    <xsl:sequence select="
        if ($n = 1 or $n = 2)
        then $acc($n)
        else
            f:fib($n - 1, function($x) {
                f:fib ($n - 2, function($y) {
                    $acc($x + $y)
                })
            })" />
</xsl:function>
```

Monads

Outline

- Introducing higher order functions
- Function items, partial function application and closures
- Efficiency techniques in XSLT 3.0 + HOF
- **Summary**

Wrapping it up

- Functions are now first class citizens
- Functions improve coding efficiency
- HOFs open a whole new range of optimization techniques


Thank you!

a distributed streaming xslt 3.0 processor ctional programming

- March 4, private beta launch
- April 15, public beta launch
- May, June, general availability release
- Q3 enterprise edition with distributed processing launch