tekom Jahrestagung 2019
STUTTGART, 12. – 14. NOVEMBER

<Tutorial>

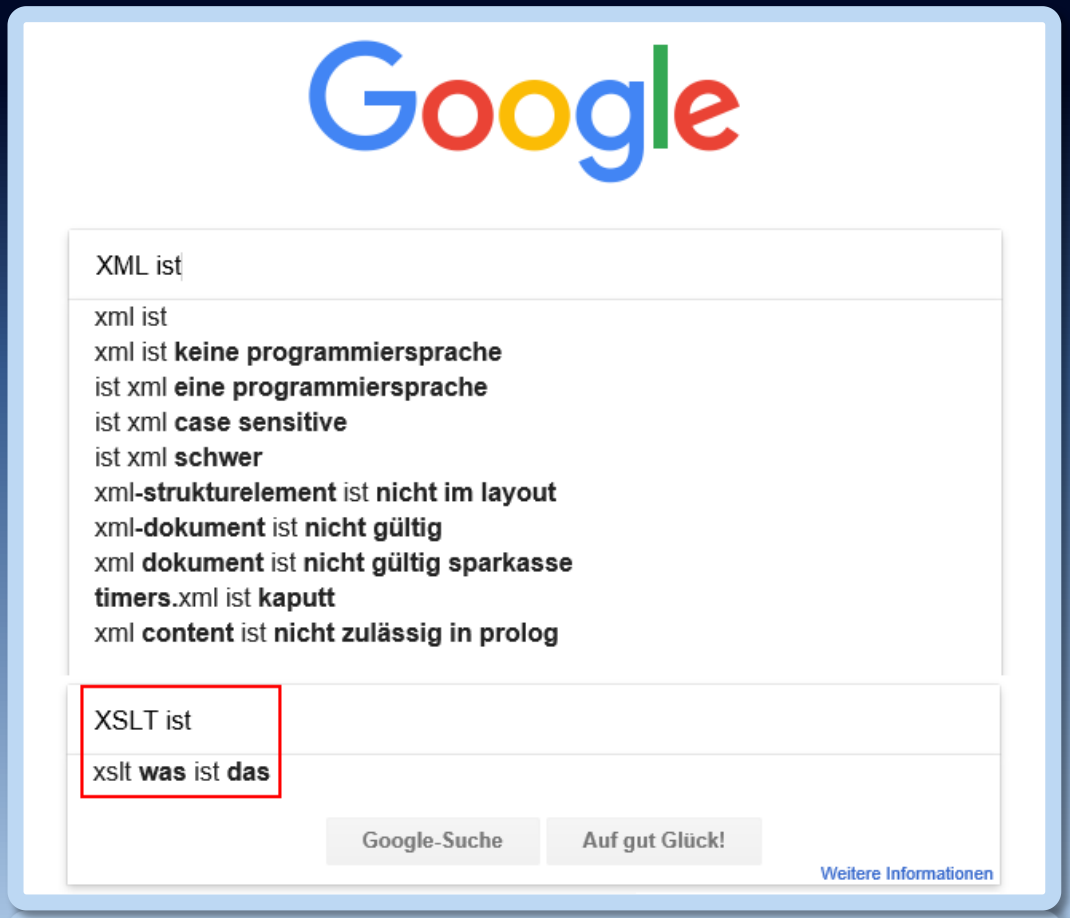
# Rechnen, Runden, Rekursion – das numerische Potenzial von XSLT


Dr. Thomas Meinike
Hochschule Merseburg

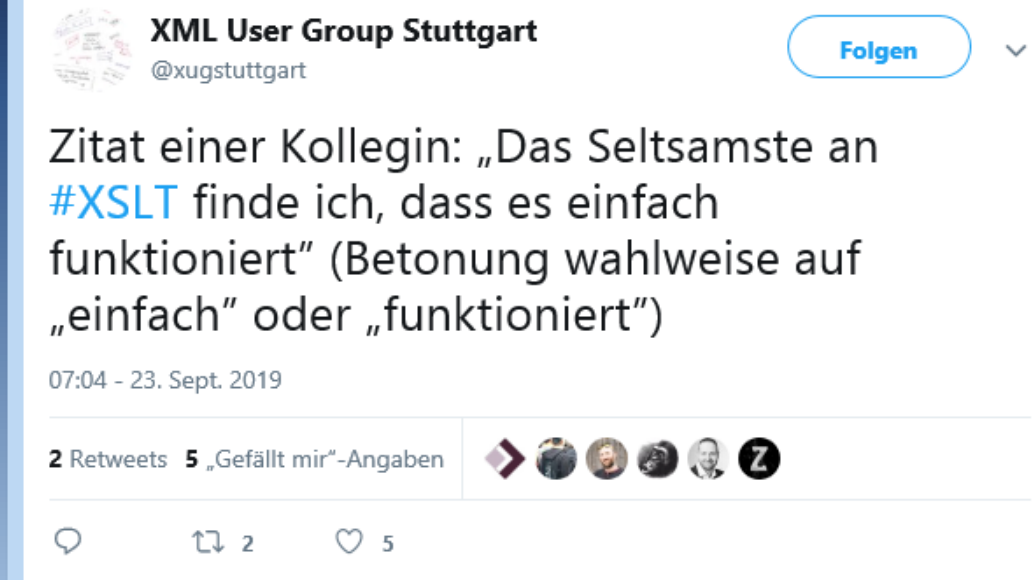
thomas.meinike@hs-merseburg.de
https://datenverdrahten.de/
@XMLArbyter


HoME
HOCHSCHULE
MERSEBURG FH
University of
Applied Sciences

14. November 2019 / Stuttgart


Google
XML ist
xml ist
xml ist keine programmiersprache
ist xml eine programmiersprache
ist xml case sensitive
ist xml schwer
xml-strukturelement ist nicht im layout
xml-dokument ist nicht gültig
xml dokument ist nicht gültig sparkasse
timers.xml ist kaputt
xml content ist nicht zulässig in prolog
XSLT ist
xslt was ist das
Google-Suche
Auf gut Glück!
Weitere Informationen

# 0. Einführung



# 0. Einführung

## XSLT und XPath 1/3

- **XSLT** wird zur Transformation von XML (und ggf. anderen textbasierten Formaten) in typische Ausgabeformate (XML, HTML und Text) verwendet

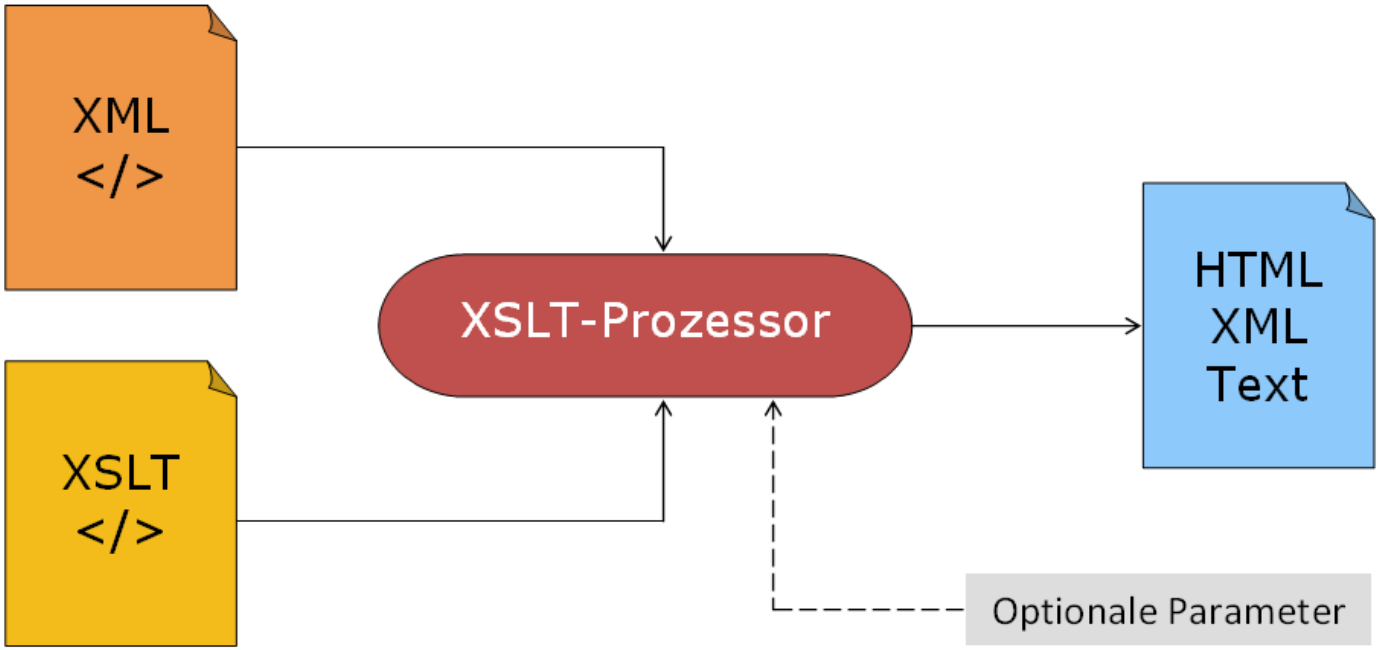
- **XPath** dient zur Abfrage von Knoten und ihren Inhalten und bietet zusätzliche Funktionalitäten

- Versionsgeschichte:
  - ✓ XSLT (16.11.2019 = 20 Jahre 🎉)
    - 11/1999: 1.0
    - 01/2007: 2.0
    - 06/2017: 3.0
  - ✓ XPath (ab 2.0 auch Untermenge von XQuery)
    - 11/1999: 1.0
    - 01/2007: 2.0
    - 03/2017: 3.1

- Vielfältige Erweiterungen in den Versionen > 1.0, speziell die überwiegend numerisch orientierten, sollen hier im Mittelpunkt stehen

## XSLT und XPath 2/3

- Verarbeitungsprinzip:




- Populärer und leistungsstarker Prozessor bis XSLT 3.0 / XPath 3.1: **Saxon (HE / PE / EE)** von Michael Kay (Saxonica.com), Editor der XSLT-Spezifikationen

- java -jar saxon9he.jar -s:name.xml -xsl:name.xsl -o:name.ext

## XSLT und XPath 3/3

- Grundgerüst:
  - ✓ XSLT-Stylesheet, hier für Version 3.0, mit den typischen Namensräumen für die im Weiteren verwendeten Techniken

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="3.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:map="http://www.w3.org/2005/xpath-functions/map"
  xmlns:math="http://www.w3.org/2005/xpath-functions/math"
  xmlns:array="http://www.w3.org/2005/xpath-functions/array"
  xmlns:tm="http://www.datenverdrahten.de/tm"
  exclude-result-prefixes="#all">

  <!-- weitere Inhalte ... -->

</xsl:stylesheet>
```

# 1. Template-Mechanismus

**Geoff Tidey**
@GeoffTidey

Folgen

having xslt fun today
<xsl:template match="lunch">
 <bread>
  <xsl:call-template name="bacon"/>
 </bread>
</xsl:template>
Output: Nom Nom

04:16 - 5. Apr. 2011

**1** Retweet

1

# 1. Template-Mechanismus

## Templates bilden die funktionale Basis 1/3

- **xsl:template** mit @match + **xsl:apply-templates**, ggf. mit @select:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wurzel>
  <kind bsp="a">Inhalt 1</kind>
  <kind bsp="b">Inhalt 2</kind>
  <kind bsp="c">Inhalt 3</kind>
</wurzel>
```

```xml
<xsl:template match="wurzel">
  <output>
    <xsl:apply-templates/>
  </output>
</xsl:template>

<xsl:template match="kind">
  <neu>
    <xsl:value-of select=
    "fn:concat(., ' | ', @bsp)"/>
  </neu>
</xsl:template>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<output>
  <neu>Inhalt 1 | a</neu>
  <neu>Inhalt 2 | b</neu>
  <neu>Inhalt 3 | c</neu>
</output>
```

# 1. Template-Mechanismus

## Templates bilden die funktionale Basis 2/3

- Benannte Templates mit **xsl:call-template** + @name + xsl:with-param / xsl:param (rekursiv aufrufen):

```
<xsl:template match="wurzel">
  ...
  <p>Bewertung:
    <xsl:call-template name="sterne">
      <xsl:with-param name="s" select="sterne"/>
    </xsl:call-template>
  </p>
  ...
</xsl:template>

<xsl:template name="sterne">
  <xsl:param name="s"/>
  <xsl:if test="$s > 0">
    <img src="stern.png" alt="*"/>
    <xsl:call-template name="sterne">
      <xsl:with-param name="s" select="$s - 1"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>
```

<-----

```
<?xml …?>
<wurzel>
  <sterne>5</sterne>
</wurzel>
```

->

```
<p>Bewertung:
  <img src="stern.png" alt="*"/>
  <img src="stern.png" alt="*"/>
  <img src="stern.png" alt="*"/>
  <img src="stern.png" alt="*"/>
  <img src="stern.png" alt="*"/>
</p>
```

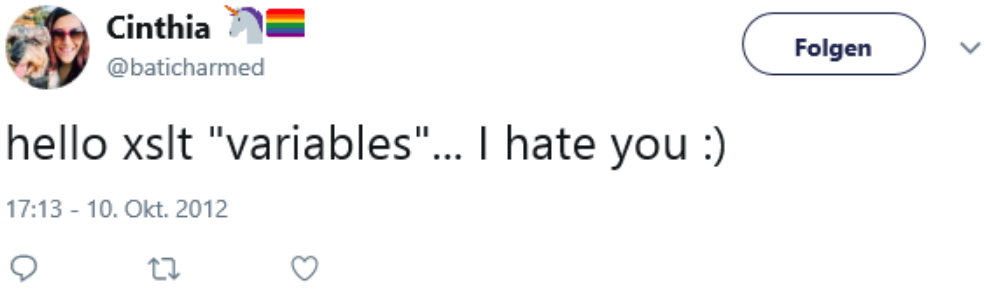
Bewertung: ★★★★★

## Templates bilden die funktionale Basis 3/3

- Nützliches Feature von XSLT 3.0: mit **xsl:initial-template** benanntes Template:
  - ✓ kann als Startpunkt einer komplexeren Transformation dienen
  - ✓ ermöglicht Transformationen ohne explizite XML-Zuweisung (bisher wurde dazu üblicherweise ein leeres Dokument wie <root/> verwendet)
  - ✓ somit lassen sich z. B. Tests von Funktionen kompakt ausführen
  - ✓ praktische Relevanz auch für Anwendungen mit Saxon-JS

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="3.0" ...>

  <xsl:output .../>

  <xsl:template name="xsl:initial-template">
    <!-- weitere Operationen ... -->
  </xsl:template>

</xsl:stylesheet>
```

# 2. Variablenkonzept



# 2. Variablenkonzept

## Variablen und Parameter 1/3

- Variablen sind im Grunde Konstanten, d. h. nach Deklaration nicht mehr änderbar jedoch zyklisch im Kontext von **Templates** oder **xsl:for-each**-Konstrukten neu erzeugbar
- Variablen und Parameter sind am vorangestellten **$**-Zeichen identifizierbar (**$name**)
- Parameter werden entweder von außen an ein Stylesheet übergeben oder im Kontext von **xsl:call-template** verwendet (siehe Template-Mechanismus)
- Auf oberster Ebene lassen sich globale Variablen und Parameter erzeugen, ansonsten sind diese auf ihre lokale Umgebung beschränkt. Ab 2.0 können Schema-Datentypen über das **as**-Attribut zugewiesen werden
- Neben Einzelwerten sind auch Sequenzen von Werten oder XML-Teilstrukturen adressierbar
- Benennung nach den XML-Namensregeln, also $name1, aber nicht $1name

# 2. Variablenkonzept

## Variablen und Parameter 2/3

- Beispiele für Zuweisungen und Abfragen:

```
<!-- Grundtyp Number -->
<xsl:variable name="var1" select="23"/>

<!-- Grundtyp String -->
<xsl:variable name="var2" select="'Text'"/>

<!-- XML-Schema-Datentypen ab 2.0 -->
<xsl:variable name="var3" select="42" as="xs:integer"/>

<xsl:variable name="var4" select="6.66" as="xs:decimal"/>

<xsl:variable name="var5" select="'Hallo Welt!'" as="xs:string"/>
```

```
<xsl:value-of select="$var3 div 2"/> <!-- 21 -->

<xsl:value-of select="$var4 * 100"/> <!-- 666 -->

<xsl:value-of select="$var5"/> <!-- Hallo Welt! -->
```

# 2. Variablenkonzept

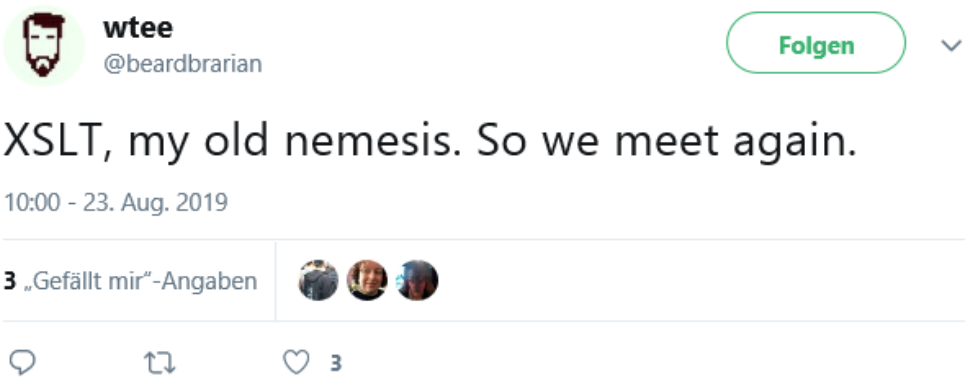
## Variablen und Parameter 3/3

- Beispiele für Zuweisungen und Abfragen:

```
<!-- Sequenz -->
<xsl:variable name="var6" select="(10, 20, 30, 40, 50)" as="xs:integer*"/>

<!-- XML-Dokumentfragment -->
<xsl:variable name="var7" as="document-node()">
  <xsl:document>
    <name1>
      <name2>Inhalt</name2>
    </name1>
  </xsl:document>
</xsl:variable>
```

```
<xsl:value-of select="$var6[3]"/> <!-- 30 -->

<xsl:value-of select="$var7/name1/name2"/> <!-- Inhalt -->
```

# 3. Grundrechenarten



# 3. Grundrechenarten

## Sind seit XSLT / XPath 1.0 verfügbar

- Operatoren:
  - ✓ Addition (+), Subtraktion (–), Multiplikation (*)
  - ✓ Division (**div**), übliches Zeichen / wird für XML-Abfragen wie a/b/c verwendet
  - ✓ Modulo-Division mit **mod**, ermittelt den Rest einer ganzzahligen Division
  - ✓ **idiv** ermittelt den ganzzahligen Anteil einer Division (ab XPath 2.0)
- Beispiele (analog mit Variablen wie **$var1** + **$var2** usw.):

```
<xsl:value-of select="10 * (100 div (30 - 5)) + 2"/> <!-- 42 -->

<xsl:value-of select="10 mod 3"/> <!-- 1 -->

<xsl:value-of select="10 mod 2"/> <!-- 0 -->

<xsl:value-of select="15 idiv 4"/> <!-- 3.75 ~> 3 -->
```

# 4. Logiktests

# 4. Logiktests

## Prüfungen bei Knoten- und Wertabfragen

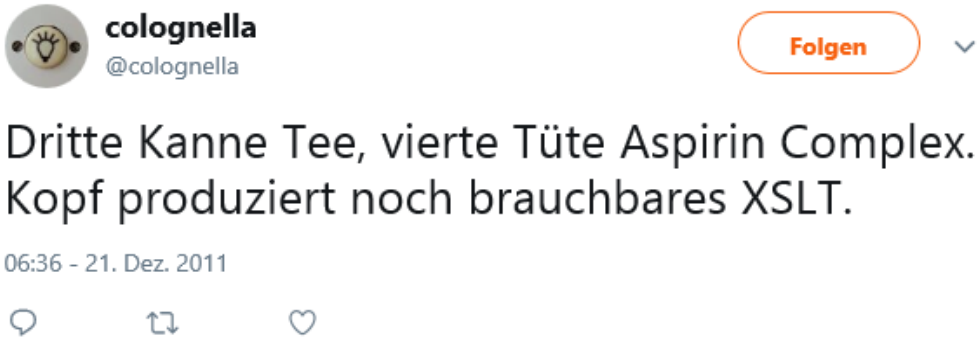
- UND → **and** | ODER → **or** | Negation → **not()**
- Einsatz in XPath-Ausdrücken, u. a. beim **test**-Attribut von **xsl:if** und **xsl:when**
- Vergleichsoperatoren:
  - ✓ < → < | > → > | = gleich | <= → <= | >= → >= | != ungleich
  - ✓ ab 2.0 auch symbolisch: lt, gt, eq, le, ge, ne (eq + e: equal, l: less, g: greater, n: not)

```
<xsl:variable name="a" select="23"
  as="xs:integer"/>
<xsl:variable name="b" select="42"
  as="xs:integer"/>
```

```
<xsl:if test="$a lt $b">
  <xsl:value-of
    select="'Test bestanden!'"/>
  <!-- weiterer Code ... -->
</xsl:if>
```

```
<xsl:choose>
  <xsl:when test="$a lt $b">
    <xsl:value-of select="'Test
      bestanden!'"/>
    <!-- weiterer Code ... -->
  </xsl:when>
  <xsl:otherwise>
    <!-- weiterer Code für den
         alternativen Fall ... -->
  </xsl:otherwise>
</xsl:choose>
```

# 5. Runden


cologne lla
@cologne lla
Folgen
Dritte Kanne Tee, vierte Tüte Aspirin Complex.
Kopf produziert noch brauchbares XSLT.
06:36 - 21. Dez. 2011

# 5. Runden

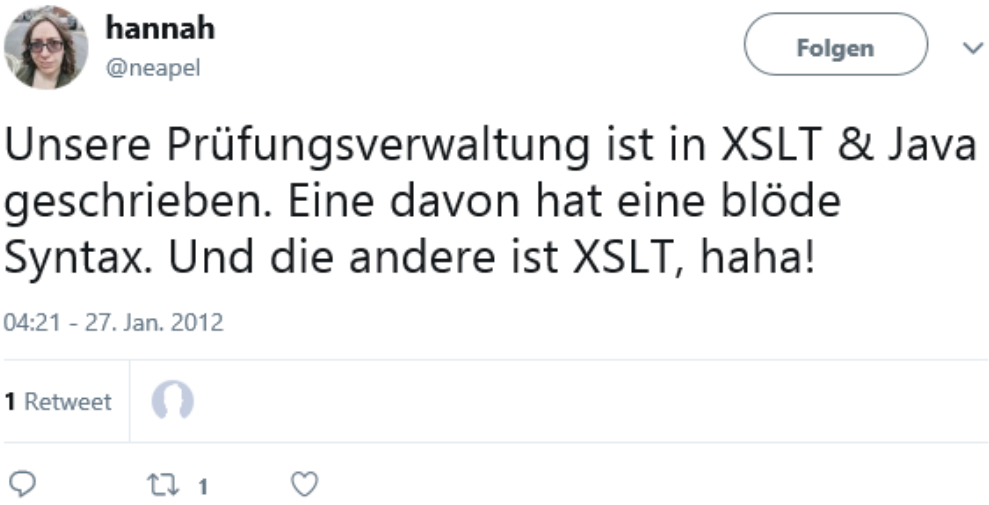
## (Auf-/Ab-)Runden, ggf. mit Vorgabe von Nachkommastellen

- fn:round() | fn:ceiling() | fn:floor() | fn:round-half-to-even() [2.0]:

```
<xsl:variable name="pi" select="math:pi()" as="xs:double"/>


<xsl:value-of select="$pi"/> <!-- 3.141592653589793 -->

<xsl:value-of select="fn:round($pi)"/> <!-- 3 -->

<xsl:value-of select="fn:round($pi, 2)"/> <!-- 3.14 -->

<xsl:value-of select="fn:ceiling($pi)"/> <!-- 4 -->

<xsl:value-of select="fn:floor($pi)"/> <!-- 3 -->

<xsl:value-of select="fn:round-half-to-even($pi, 2)"/> <!-- 3.14 -->
```

- »fn:round and fn:round-half-to-even produce the same result in all cases except when the argument is exactly midway between two values with the required precision.« [https://www.w3.org/TR/xpath-functions-31/]

# 6. Aggregatfunktionen



# 6. Aggregatfunktionen

## Ermittlung von Minimum, Maximum, Mittelwert, Summe und Anzahl auf numerischen Knoten- oder Wertesequenzen

- fn:sum() und fn:count() seit XPath 1.0 | fn:min(), fn:max(), fn:avg() ab 2.0
- Beispiel einer Spannungs-/Strom-Messung mit Berechnung des Widerstandes (**spannung** div **strom**) und aggregierter Werte bezogen auf die Spannung:

```
<?xml … ?>
<messdaten ...>
  <messung>
    <spannung>25</spannung>
    <strom>0.10</strom>
  </messung>
  ...
</messdaten>
```

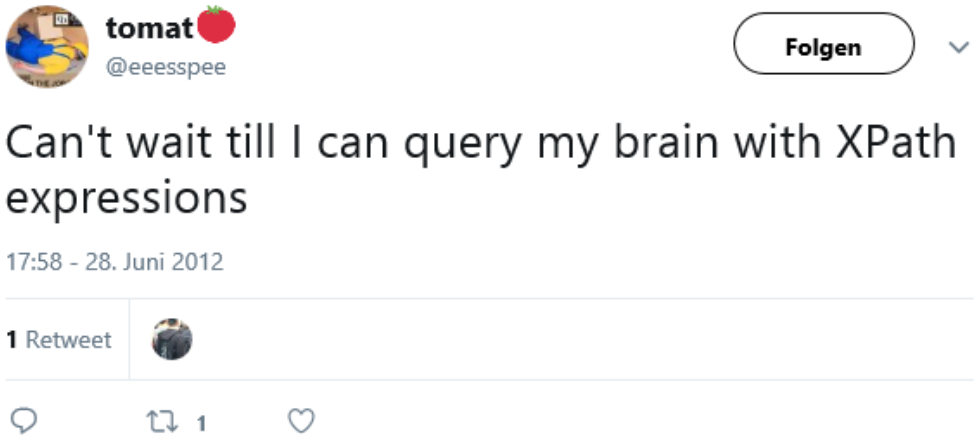
**Messergebnisse**

| Nr. | U [V] | I [A] | R [Ω] |
|---|---|---|---|
| 1 | 25 | 0.10 | 250 |
| 2 | 60 | 0.20 | 300 |
| 3 | 110 | 0.31 | 355 |
| 4 | 155 | 0.36 | 431 |
| 5 | 200 | 0.40 | 500 |
| 6 | 230 | 0.42 | 548 |

$U_{min}$ = 25 V | $U_{max}$ = 230 V | $U_{sum}$ = 780 V | $U_{avg}$ = 130 V | n = 6

```
U<sub>min</sub> = <xsl:value-of select="fn:min(messung/spannung)"/> V |
U<sub>max</sub> = <xsl:value-of select="fn:max(messung/spannung)"/> V |
U<sub>sum</sub> = <xsl:value-of select="fn:sum(messung/spannung)"/> V |
U<sub>avg</sub> = <xsl:value-of select="fn:avg(messung/spannung)"/> V |
n = <xsl:value-of select="fn:count(messung/spannung)"/>
```

# 7. Zahlenformatierung



# 7. Zahlenformatierung

## Mit fn:format-number() und xsl:number nach spez. Kriterien

- **fn:format-number()**:

```
<xsl:value-of select="fn:format-number(123456, '#.00')"/> <!-- 123456.00 -->

<xsl:value-of select="fn:format-number(1.2345, '#.00')"/> <!-- 1.23 -->

<xsl:value-of select="fn:format-number(123456, '###,###.00')"/> <!-- 123,456.00 -->

<xsl:value-of select="fn:format-number(0.1234, '#%')"/> <!-- 12% -->
```

- **xsl:number**:

```
<xsl:number value="123456" grouping-separator="/" grouping-size="2"/>
<!-- 12/34/56 -->

<xsl:number value="23" format="I"/> <!-- XXIII -->

<xsl:number value="42" format="w" ordinal="yes" lang="de"/>
<!-- forty-second (Saxon-HE) | zweiundvierzigste (Saxon-PE/EE) -->
```

# 8. Umgang mit Datum und Zeit

**Peter Cooper Jr.**
@petercooperjr

Folgen

Dealing with date/time info in XSLT can be a bit of a pain.

10:32 - 25. Juni 2012

1

# 8. Umgang mit Datum und Zeit

## Abfragen und Formatieren nach Mustern (ab 2.0)

- **fn:current-date() | fn:current-time() | fn:current-dateTime() | fn:format-…()**:

```
<xsl:variable name="d" select="fn:current-date()" as="xs:date"/>

<xsl:variable name="t" select="fn:current-time()" as="xs:time"/>

<xsl:variable name="dt" select="fn:current-dateTime()" as="xs:dateTime"/>
```

```
<xsl:value-of select="$d"/> <!-- 2019-10-26+02:00 | noch Sommerzeit -->

<xsl:value-of select="$t"/> <!-- 13:37:01.441+02:00 -->

<xsl:value-of select="$dt"/> <!-- 2019-10-26T13:37:01.441+02:00 -->

<xsl:value-of select="fn:format-date($d, '[D01].[M01].[Y0001]')"/>
                                                    <!-- 26.10.2019 -->

<xsl:value-of select="fn:format-time($t, '[H01]:[m01]')"/> <!-- 13:37 -->
```

# 8. Umgang mit Datum und Zeit

## Rechnen mit Datums-/Zeitwerten (ab 2.0)

- Anzahl der Tage bis Weihnachten:

```
<xsl:variable name="w" select="xs:date('2019-12-24')" as="xs:date"/>

<xsl:value-of select="fn:days-from-duration($w - $d)"/> <!-- 59 Tage -->
```

- Datum in 100 Tagen (P1D steht für Period 1 Day, auch P1Y5M usw.):

```
<xsl:variable name="d_pl100" select="$d + 100 * xs:dayTimeDuration('P1D')"
              as="xs:date"/>

<xsl:value-of select="fn:format-date($d_pl100, '[D01].[M01].[Y0001]')"/>
                                                    <!-- 03.02.2020 -->
```

- Hinweis: am 14.11.2019 sind es noch 40 Tage bis Weihnachten und in 100 Tagen liegt der 22.02.2020

# 9. Sortieren, Gruppieren und Filtern

## Kompakte Zusammenfassung 1/3

- **Sortieren** mit **xsl:sort**:
  - ✓ unterhalb von xsl:apply-templates, xsl:for-each, xsl:for-each-group
  - ✓ nach Datentyp: data-type="number|**text**" (fett = default)
  - ✓ nach Richtung: order="**ascending**|descending" (fett = default)
  - ✓ mehrfache Sortierung mit Abfolge von xsl:sort ist möglich
  - ✓ wird u. a. im bereits gezeigten Aggregat-Beispiel Messung angewendet:

```
<xsl:apply-templates select="messung">
  <xsl:sort select="spannung" data-type="number" order="ascending"/>
</xsl:apply-templates>
```

# 9. Sortieren, Gruppieren und Filtern

## Kompakte Zusammenfassung 2/3

- **Gruppieren** mit **xsl:for-each-group** (ab 2.0):
  - ✓ mit den Attributen **select** und **group-by**
  - ✓ Verarbeitung mit **fn:current-group()** + **fn:current-grouping-key()**
  - ✓ siehe Beispiel Technologien:

```
<xsl:for-each-group select="technologie" group-by="herausgeber">
  <xsl:sort select="herausgeber" data-type="text" order="ascending"/>
  <h2><xsl:value-of select="fn:current-grouping-key()"/></h2>
  <ul>
    <xsl:for-each select="fn:current-group()">
      <xsl:sort select="bezeichnung" data-type="text" order="ascending"/>
      <li>
        <xsl:value-of select="fn:concat(bezeichnung, ' (', kurzbezeichnung,
          ', ', herausgeber/@jahr, ')')"/>
      </li>
    </xsl:for-each>
  </ul>
</xsl:for-each-group>
```

**W3C**

- Extensible Hypertext Markup Language (XHTML, 2000)
- Extensible Stylesheet Language Transformations (XSLT, 1999)
- Mathematical Markup Language (MathML, 1998)
- Scalable Vector Graphics (SVG, 2001)

# 9. Sortieren, Gruppieren und Filtern

## Kompakte Zusammenfassung 3/3

- **Filtern**:
  - ✓ **fn:distinct-values()**:

```
<xsl:variable name="seq" select="1, 2, 3, 4, 5, 1, 2, 23, 42" as="xs:integer*"/>

<xsl:value-of select="fn:distinct-values($seq)"/> <!-- 1 2 3 4 5 23 42 -->
```

  - ✓ **fn:contains() | fn:starts-with() | fn:ends-with() | fn:string-length()**:

```
<xsl:variable name="str" select="'ABCDEF'"/>

<xsl:value-of select="fn:contains($str, 'CD')"/>     <!-- true -->
<xsl:value-of select="fn:starts-with($str, 'ABC')"/> <!-- true -->
<xsl:value-of select="fn:ends-with($str, 'E')"/>     <!-- false -->
<xsl:value-of select="fn:string-length($str)"/>      <!-- 6 -->
```

  - ✓ **Weitere Ansätze**:
    - fn:matches() + xsl:analyze-string → reguläre Ausdrücke
    - some/every … in … satisfies → Primzahlen unter HOF
    - Mengenoperationen auf Sequenzen: except, intersect, union […]

# 10. Reguläre Ausdrücke

# 10. Reguläre Ausdrücke

## Aufbau von Zeichenketten prüfen

- **xsl:analyze-string** + **xsl:matching-substring** | **xsl:non-matching-substring**

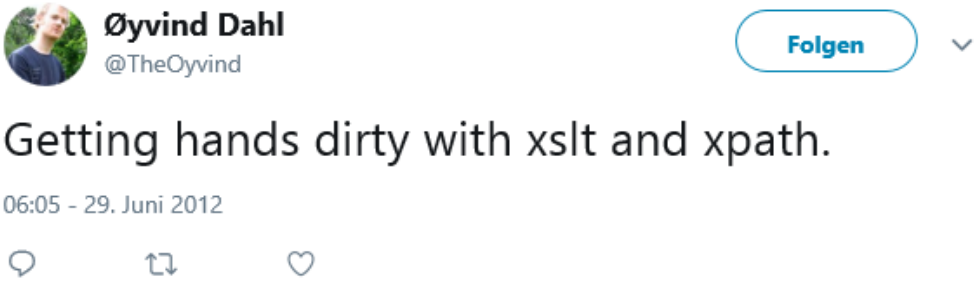
  am Beispiel Postleitzahl: (D )?([0-9]{5}) bzw. (D )?([\d]{5}):

```
<xsl:variable name="test" select="'D 06217'"/> <!-- PLZ -->

<xsl:analyze-string select="$test" regex="(D )?(\d{{5}})">
  <xsl:matching-substring>
    <xsl:text>PLZ ok | </xsl:text>
    <xsl:text>Landeskennung: </xsl:text><xsl:value-of select="regex-group(1)"/>
    <xsl:text> | Zahlencode: </xsl:text><xsl:value-of select="regex-group(2)"/>
  </xsl:matching-substring>
  <xsl:non-matching-substring>
    <xsl:text>fehlerhafte PLZ</xsl:text>
  </xsl:non-matching-substring>
</xsl:analyze-string>
<!-- PLZ ok | Landeskennung: D | Zahlencode: 06217 -->
```

- Einzelprüfung über Funktion **fn:matches**(string, regex, flags?):

```
<xsl:value-of select="fn:matches('Ein Test', 'TEST', 'i')"/>
<!-- true | i = case-insensitive, also Test, TEST, test usw. möglich -->
```

# 11. Erweiterte Inline-Abfragen



# 11. Erweiterte Inline-Abfragen

## Logik innerhalb von Attribut- oder Wertausgaben 1/2

- **if** … **then** … **else** | Beispiel BMI-Berechnung:

```
<xsl:variable name="akt_bmi" select="tm:BMI(gewicht, groesse)"/>
<td class="{if($akt_bmi lt 16) then 'sun' else
  if($akt_bmi ge 16 and $akt_bmi lt 17) then 'mun' else
  if($akt_bmi ge 17 and $akt_bmi lt 18.5) then 'lun' else
  if($akt_bmi ge 18.5 and $akt_bmi lt 25) then 'nog' else
  if($akt_bmi ge 25 and $akt_bmi lt 30) then 'pad' else
  if($akt_bmi ge 30 and $akt_bmi lt 35) then 'ad1' else
  if($akt_bmi ge 35 and $akt_bmi lt 40) then 'ad2' else
  if($akt_bmi ge 40) then 'ad3' else()}">
  <xsl:value-of select="$akt_bmi"/>
</td>
```

```
<xsl:function name="tm:BMI" as="xs:decimal">
  <xsl:param name="ge" as="xs:decimal"/>
  <xsl:param name="gr" as="xs:decimal"/>
  <xsl:value-of select="fn:round-half-to-even
    ($ge div ($gr * $gr), 2)"/>
</xsl:function>
```

```
.nog /* Normalgewicht */ {
  color: #000; background-color: #7CFC7C;
}
```

| Name | Gewicht / kg | Größe / m | BMI |
|---|---|---|---|
| A | 40 | 1.60 | 15.62 |
| B | 60 | 1.75 | 19.59 |
| C | 85 | 1.70 | 29.41 |
| D | 90 | 1.65 | 33.06 |
| E | 55 | 1.85 | 16.07 |
| F | 110 | 1.50 | 48.89 |

# 11. Erweiterte Inline-Abfragen

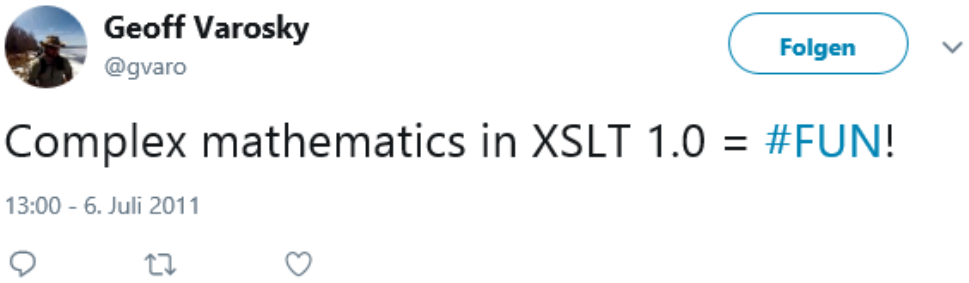
## Logik innerhalb von Attribut- oder Wertausgaben 2/2

- **for** … **in** … **return** | Beispiel Warenkorb:

```
<?xml version="1.0" encoding="UTF-8"?>
<bestellungen>
  <auswahl>
    <produkt>A</produkt> <anzahl>2</anzahl> <preis>14.50</preis>
  </auswahl>
  <auswahl>
    <produkt>B</produkt> <anzahl>5</anzahl> <preis>9.20</preis>
  </auswahl>
  <auswahl>
    <produkt>C</produkt> <anzahl>1</anzahl> <preis>39.00</preis>
  </auswahl>
</bestellungen>
```

```
<xsl:template match="bestellungen">
  <xsl:variable name="gesamt" select="fn:sum(
    for $a in auswahl return $a/anzahl * $a/preis)" as="xs:double"/>

  <xsl:value-of select="fn:concat('Summe[anzahl * preis] = ', $gesamt)"/>
                                                        <!-- 114 -->
</xsl:template>
```

# 12. Mathematische Funktionen



# 12. Mathematische Funktionen

## XPath 3.1 stellt 14 vordefinierte math-Funktionen bereit 1/2

- xmlns:math="http://www.w3.org/2005/xpath-functions/math"
- Beispielausgabe von **math.xsl**:

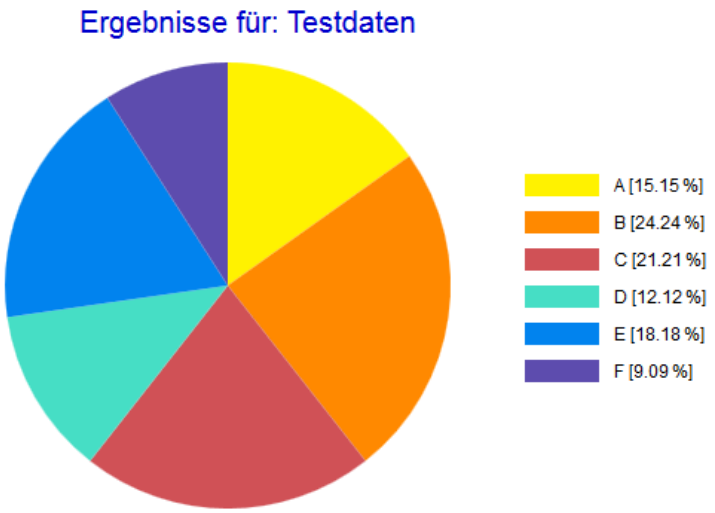
| Trigonometrische Funktionen und π | |
|---|---|
| **math:acos(0.5)** | 1.0471975511965979 |
| **math:asin(0.5)** | 0.5235987755982989 |
| **math:atan(0.5)** | 0.4636476090008061 |
| **math:atan2(1, 0.5)** | 1.1071487177940904 |
| **math:cos(0.5)** | 0.8775825618903728 |
| **math:sin(0.5)** | 0.479425538604203 |
| **math:tan(0.5)** | 0.5463024898437905 |
| **math:pi()** | 3.141592653589793 |
| Exponential- und Logarithmus-Funktionen | |
| **math:exp(2)** [= e^2] | 7.38905609893065 |
| **math:exp10(4)** [= 10^4] | 10000 |
| **math:log(5)** [= ln(5)] | 1.6094379124341003 |
| **math:log10(100)** [= lg(100)] | 2 |
| Potenz- und Wurzel-Funktionen | |
| **math:pow(2, 5)** [= 2^5] | 32 |
| **math:sqrt(81)** [= √81] | 9 |

# 12. Mathematische Funktionen

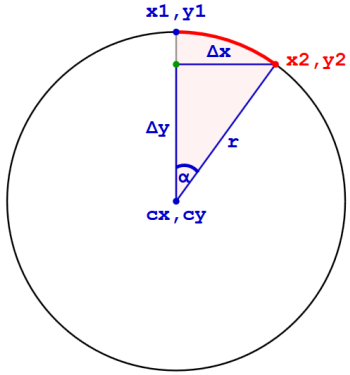
## XPath 3.1 stellt 14 vordefinierte math-Funktionen bereit 2/2

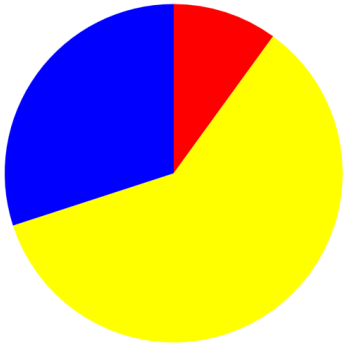
- Umsetzung eines SVG-Piecharts aus XML-Daten mit **math:sin()** | **math:cos()**:

```
<?xml ... ?>
<daten info="Testdaten">
  <satz>
    <wert>50</wert>
    <text>A</text>
    <farbe>#FFF200</farbe>
  </satz>
  ...
</daten>
```




Kreissegmente zeichnen / berechnen

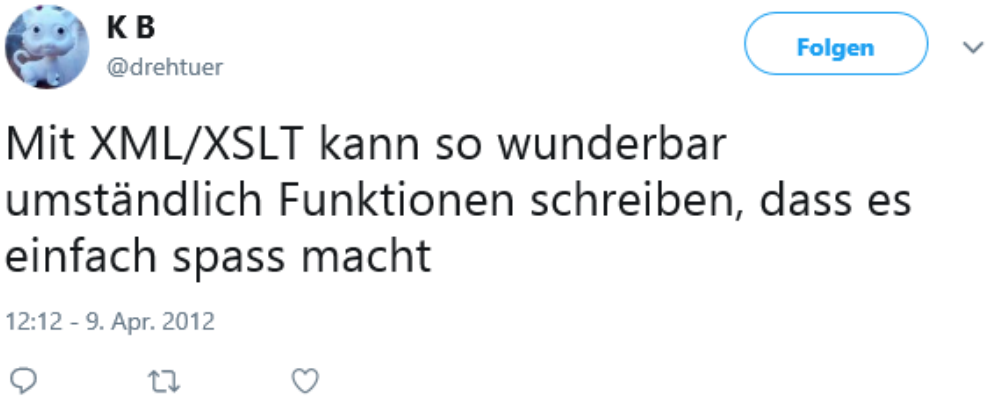


$x2 = cx + \Delta x = cx + r \cdot \sin \alpha$
$y2 = cy - \Delta y = cy - r \cdot \cos \alpha$

```
Kreissegment <= 180°
<path d="Mcx,cy Lx1,y1 Ar,r 0 0,1 x2,y2 Z" ... />

Kreissegment > 180°
<path d="Mcx,cy Lx1,y1 Ar,r 0 1,1 x2,y2 Z" ... />
```

# 13. Benutzerdefinierte Funktionen



# 13. Benutzerdefinierte Funktionen

## Deklaration mit xsl:function (ab 2.0)

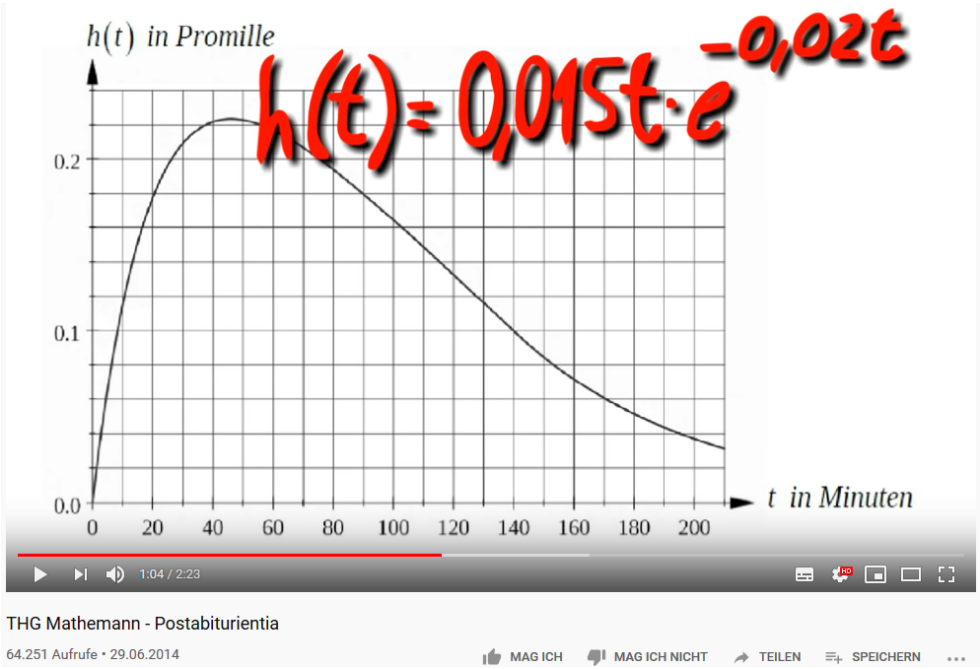
- Element(e) unterhalb von xsl:stylesheet platzieren (bzw. aus lib.xsl importieren)
- Argumente als Parameter (**xsl:param**), optionale Argumente nur über mehrfache Funktionsdeklaration möglich
- Funktionsname mit eigenem Namensraum verknüpft, z. B. **my:funcname**
- Rückgabe von atomaren Werten oder Sequenzen (xsl:value-of | xsl:sequence)
- Anwendung innerhalb von XPath-Ausdrücken wie vorgefertigte Funktionen
- Attribut override (yes|no) ermöglicht Überschreiben Prozessor-eigener Funktionen mit identischem Namen (z. B. saxon:sort)

```
<xsl:function name="my:funcname" as="xs:decimal">
  <xsl:param name="a" as="xs:decimal"/>
  <xsl:param name="b" as="xs:decimal"/>
  <xsl:value-of select="$a + $b"/>
</xsl:function>
<!-- Aufruf: my:funcname(1, 2) -> 3 -->
```

# 13. Benutzerdefinierte Funktionen

## Beispiel THG Mathemann – Promille-Kurve 1/2

- h(t) = 0.015 t e ^ (-0.02 t) | Kurve bei 1:04 min:



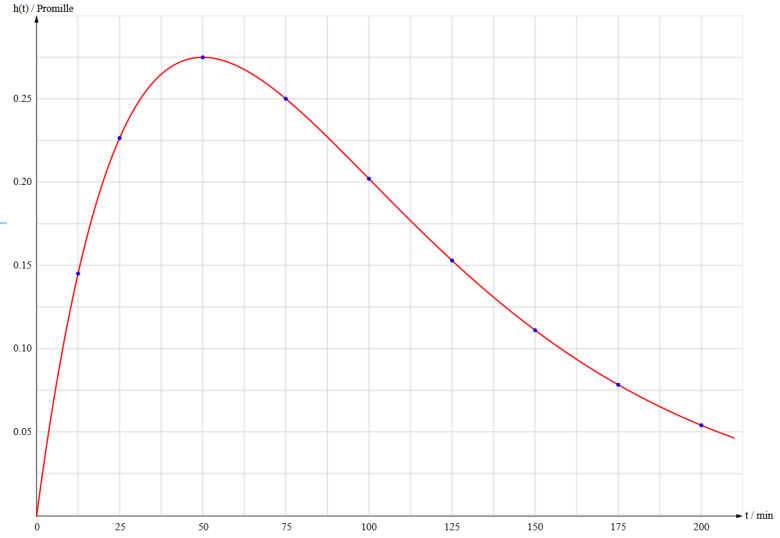

https://youtu.be/386obAbrA30

```
<xsl:function name="tm:Promille" as="xs:decimal">
  <xsl:param name="t" as="xs:decimal"/>
  <xsl:value-of select="0.015 * $t * math:exp(-0.02 * $t)"/>
</xsl:function>
```

## Beispiel THG Mathemann – Promille-Kurve 2/2

- Umsetzung als Polyline in SVG über **tm:Promille()**:

```
<polyline stroke="#F00" stroke-width="2" fill="none">
  <xsl:attribute name="points">
    <xsl:for-each select="0 to 210">
      <xsl:variable name="x" select="."/>
      <xsl:variable name="y" select="tm:Promille($x)"/>
      <xsl:value-of select="fn:concat(
        $x * 5, ',', $scale_y - fn:round-half-to-even($y * $scale_x, 2))"/>
      <xsl:if test="fn:position() != fn:last()">
        <xsl:text> </xsl:text>
      </xsl:if>
    </xsl:for-each>
  </xsl:attribute>
</polyline>
```




→ Einzelne Punkte als SVG-Kreise
u. a. Maximum bei t = 50 | h(t) = 0.276

# 13. Benutzerdefinierte Funktionen

## Beispiel OpenStreetMap-Karte 1/2

- Code für #tekom18-Vortrag zu JSON-Verarbeitung mit XSLT entwickelt
- Zur Ausgabe einer Karte in HTML via iframe oder object wird die Bounding-Box benötigt, siehe **getBBox.xsl**

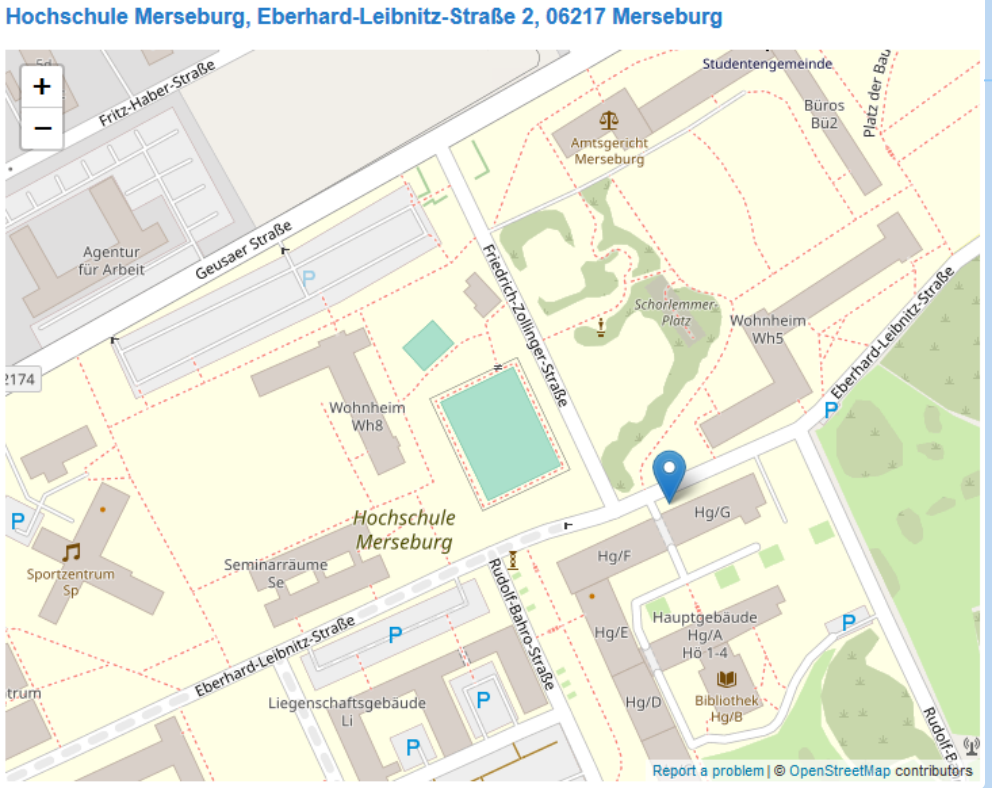
```
<!-- Geodaten für Hochschule Merseburg: -->
<xsl:variable name="lat" select="51.343895" as="xs:double"/>
<xsl:variable name="lon" select="11.976492" as="xs:double"/>

<!-- Zoomfaktor und Ausgabemaße: -->
<xsl:variable name="zoom" select="17" as="xs:integer"/>
<xsl:variable name="width" select="720" as="xs:integer"/>
<xsl:variable name="height" select="540" as="xs:integer"/>

<!-- Bounding Box via XSLT-Funktion tm:getBBox(): -->
<xsl:variable name="bbox" select="tm:getBBox($lat, $lon, $zoom, $width, $height)"
  as="xs:string"/>

<!-- object-Element zusammensetzen und ausgeben: -->
<object data="https://www.openstreetmap.org/export/embed.html?bbox={$bbox}
  &amp;marker={$lat},{$lon}&amp;layer=mapnik" width="{$width}" height="{$height}">
 <!-- Alternativ-Link -->
</object>
```

# 13. Benutzerdefinierte Funktionen

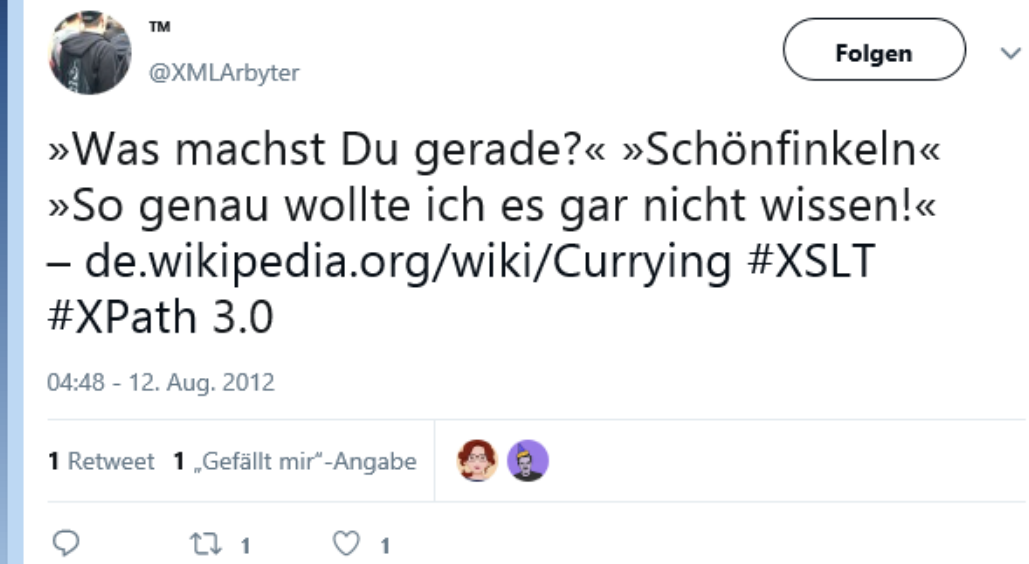
## Beispiel OpenStreetMap-Karte 2/2

- HTML-Ergebnis:

```
<object data="https://www.openstreetmap.org/export/embed.html?bbox=11.971235275268555,
  51.346148068859584,11.978960037231445,51.342529180898765&amp;marker=51.343895,11.976492
  &amp;layer=mapnik" width="720" height="540">
  <!-- Alternativ-Link -->
</object>
```



# 14. Funktionen höherer Ordnung



# 14. Funktionen höherer Ordnung

## Verwenden Werte und Funktionen als Argumente (ab 3.0)

- Anonyme Inline-Funktionen mit **function(…) {…}**: HOF nur mit Saxon-PE/EE!

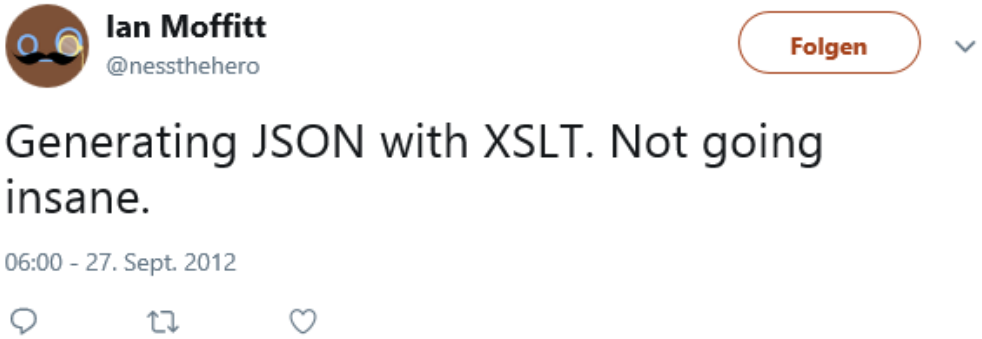
```
<!-- einfache Addition zweier Argumente: -->
<xsl:variable name="addiere" select="function($a, $b) { $a + $b }"/>

<xsl:value-of select="$addiere(23, 19)"/> <!-- 42 -->
```

- Verwendung zusätzlicher Funktionen, hier **fn:fold-left()** und **fn:filter()**:

```
<!-- alternative Fakultätsberechnung mit fn:fold-left(): -->
<xsl:variable name="fact" select="function($n) {
  fn:fold-left(1 to $n, 1, function($i, $j) { $i * $j }) }"/>

<xsl:value-of select="$fact(5)"/> <!-- 120 -->
```

```
<!-- Primzahlen im Bereich 1 bis 100 (adaptiert von
     http://docs.basex.org/wiki/Higher-Order_Functions): -->
<xsl:value-of select="let $is-prime := function($x) {
  $x gt 1 and (every $y in 2 to ($x - 1) satisfies $x mod $y ne 0) }
  return fn:filter(1 to 100, $is-prime)"/>

<!-- 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 -->
```

# 15. Arrays und Maps



# 15. Arrays und Maps

## Verfügbar ab XPath 3.1, nützlich für JSON-Zugriff 1/2

- JSON-Struktur für Konferenzdaten:

```
{
  "konferenzen": [
    {
      "kid": "k01",
      "name": "tekom-Jahrestagung 2019",
      "ort": "Messe Stuttgart",
      "tage": 3,
      "datum": ["2019-11-12", "2019-11-13", "2019-11-14"],
      "infos": "https://tagungen.tekom.de/",
      "topevent": true
    },
    {
      Ggf. weitere Datensätze …
    }
  ]
}
```

# 15. Arrays und Maps

## Verfügbar ab XPath 3.1, nützlich für JSON-Zugriff 2/2

- Abfragen auf dieser Struktur: ? = Lookup-Operator

```
<xsl:variable name="json"
  select="fn:json-doc('konferenzen.json')" as="map(*)"/>

<xsl:variable name="daten" select="$json?konferenzen" as="array(*)"/>
```

```
<xsl:value-of select="map:size($daten?1)"/>                    <!-- 7 -->

<xsl:value-of select="$daten?1?name"/>    <!-- tekom-Jahrestagung 2019 -->

<xsl:value-of select="$daten?1?ort"/>            <!-- Messe Stuttgart -->

<xsl:value-of select="array:size($daten?1?datum)"/>            <!-- 3 -->

<xsl:value-of select="$daten?1?datum?3"/>              <!-- 2019-11-14 -->
```

- Für weitere Details siehe #tekom18-Vortrag zu JSON + XSLT

# 16. Zufallszahlen

# 16. Zufallszahlen

## Zufallszahlengenerator in XPath 3.1 1/2

- **fn:random-number-generator()** erzeugt eine Map mit den Keys **number**, **next** und **permute**:

```
<xsl:variable name="myrng" select="fn:random-number-generator()"
              as="map(xs:string, item())"/>
```

  - ✓ **$myrng?number** ermittelt 0 ≤ Zufallszahl < 1 als xs:double
  - ✓ **$myrng?*next*()?number** liefert eine weitere Zufallszahl
  - ✓ **$myrng?*permute*(1 to 10)** ergibt die zufällige Anordnung einer Eingabe-Sequenz, hier von 1 bis 10, z. B.: 3 7 5 4 6 9 8 1 10 2

- Unter Verwendung weiterer Techniken wie Runden auf ganze Zahlen lässt sich analog zu Math.random() in JavaScript ein Würfelspiel bauen ...

# 16. Zufallszahlen

## Zufallszahlengenerator in XPath 3.1 2/2

- Würfeln mit **fn:random-number-generator()**:

```
<xsl:variable name="w" as="xs:integer*">
  <xsl:value-of select="xs:integer(fn:floor($myrng?number * 6) + 1)"/>
  <xsl:value-of select="xs:integer(fn:floor($myrng?next()?number * 6) + 1)"/>
  <xsl:value-of select="xs:integer(fn:floor($myrng?next()?next()?number * 6) + 1)"/>
</xsl:variable>

<p>
  <xsl:for-each select="1 to fn:count($w)">
    <xsl:variable name="p" select="fn:position()"/>
    <img src="wuerfel/w{$w[$p]}.png" alt="{$w[$p]} gewürfelt"/>
  </xsl:for-each>
</p>

<p>Summe: <xsl:value-of select="fn:sum($w)"/></p>
```

- Zuweisung der Würfelgrafiken w1.png … w6.png:

Summe: 14

# Ausblick und Referenzen

# Ausblick und Referenzen (1)

- **W3C**:
  - The Extensible Stylesheet Language Family (XSL). https://www.w3.org/Style/XSL/
  - XSL Transformations (XSLT) Version 1.0. https://www.w3.org/TR/xslt-10/
  - XML Path Language (XPath) Version 1.0. https://www.w3.org/TR/xpath-10/
  - XSL Transformations (XSLT) Version 2.0. https://www.w3.org/TR/xslt20/
  - XML Path Language (XPath) 2.0 (Second Edition). https://www.w3.org/TR/xpath20/
  - XSL Transformations (XSLT) Version 3.0. https://www.w3.org/TR/xslt-30/
  - XML Path Language (XPath) 3.1. https://www.w3.org/TR/xpath-31/
- **Saxonica**:
  - Why choose Saxon? https://saxonica.com/html/products/why-saxon.html
- **Twitter**:
  - Verwendete Tweets. https://twitter.com/

# Ausblick und Referenzen (2)

- **Meinike, T.**:
  - ✓ XSLT 2.0 und XPath 2.0 für Praktiker – Neuerungen im Überblick. In: Gesellschaft für Technische Kommunikation – tekom Deutschland e.V., Tagungsband zur Jahrestagung 2007, S. 212–215. https://datenverdrahten.de/PDF/tekom2007_Wiesbaden_Meinike.pdf
  - ✓ Tutorial: XSLT-Programmierung – effektiv und schmerzfrei! In: Gesellschaft für Technische Kommunikation – tekom Deutschland e.V., Tagungsband zur Jahrestagung 2011, S. 313–315. https://datenverdrahten.de/PDF/tekom2011_OTS11_Meinike.pdf
  - ✓ 3.0-Updates von XSLT und XPath auf einen Blick. In: Gesellschaft für Technische Kommunikation – tekom Deutschland e.V., Tagungsband zur Jahrestagung 2012, S. 341–343. https://datenverdrahten.de/PDF/tekom2012_OTS3_Meinike.pdf
  - ✓ Interaktive XML-Verarbeitung im Browser mit Saxon-JS und XSLT 3.0. In: Gesellschaft für Technische Kommunikation – tekom Deutschland e.V., Tagungsband zur Jahrestagung 2017, S. 177–180. https://datenverdrahten.de/PDF/tekom2017_IN29_Meinike.pdf
  - ✓ Verarbeitung von JSON-Daten mit aktuellen XSLT-Techniken. In: Gesellschaft für Technische Kommunikation – tekom Deutschland e.V., Tagungsband zur Jahrestagung 2018, S. 123–126. https://datenverdrahten.de/PDF/tekom2018_IN22_Meinike.pdf
  - ✓ Material und Code zu diesem Tutorial. https://datenverdrahten.de/xslt3/tekom19/tekom19_xsltut.zip

# </Tutorial>

**Matthew Phillips** @matthewcp · 6. Okt.
Why did XSLT fail?

8 2

**Norman Walsh** @ndw · 15 Std.
Failed? Not in the universe I live in.

2 1 2

# Danke für Ihr Interesse! Feedback erwünscht …

- URL direkt aufrufen: https://sundt06.honestly.de/
- Oder QR-Code scannen:

- Das Bewertungstool steht Ihnen auch noch nach der Tagung zur Verfügung!