-Teil 2 -

Schwerpunkt: Templates und Kontrollstrukturen

Gliederung

XSL Transformationen - Teil 2

- Templates -

Allgemeine Informationen

- Dokument kann mehrere xsl:template Elemente enthalten
- Templates können Namen haben

```
<xsl:template name="templateEins">
</xsl:template>
<xsl:template name="templateZwei">
</xsl:template>
```

- keine direkte Verschachtelung

```
<xsl:template name="temp1">
    <xsl:template name="temp2">
    </xsl:template>
</xsl:template>
```

- wird ein Knoten durch ein Template bearbeitet, wird dieser zum aktuellen Knoten (wichtig für XPath Ausdrücke zum Traversieren)

match-Attribut

- Festlegung für welche Daten das Template anzuwenden ist
- Zuweisung eines XPath-Ausdrucks

```
match="/"                          Wurzelelement
match="Elementname"                Jedes Element vom Typ „Elementname"
match="Elementname[1]"             Das erste Element vom Typ „Elementname"
match="Parentelement/Element"      Jedes „Element", dass in einem
                                   „Parentelement" steht
match="@Attributname"              Jedes Attribut mit Namen „Attributname"
match="@*"                         Alle Attribute
match="text()"                         Alle Textknoten
```

- bei Transformation Überprüfung, ob ein Knoten auf das Muster passt, dass im match-Attribut angegeben wurde

match-Attribut

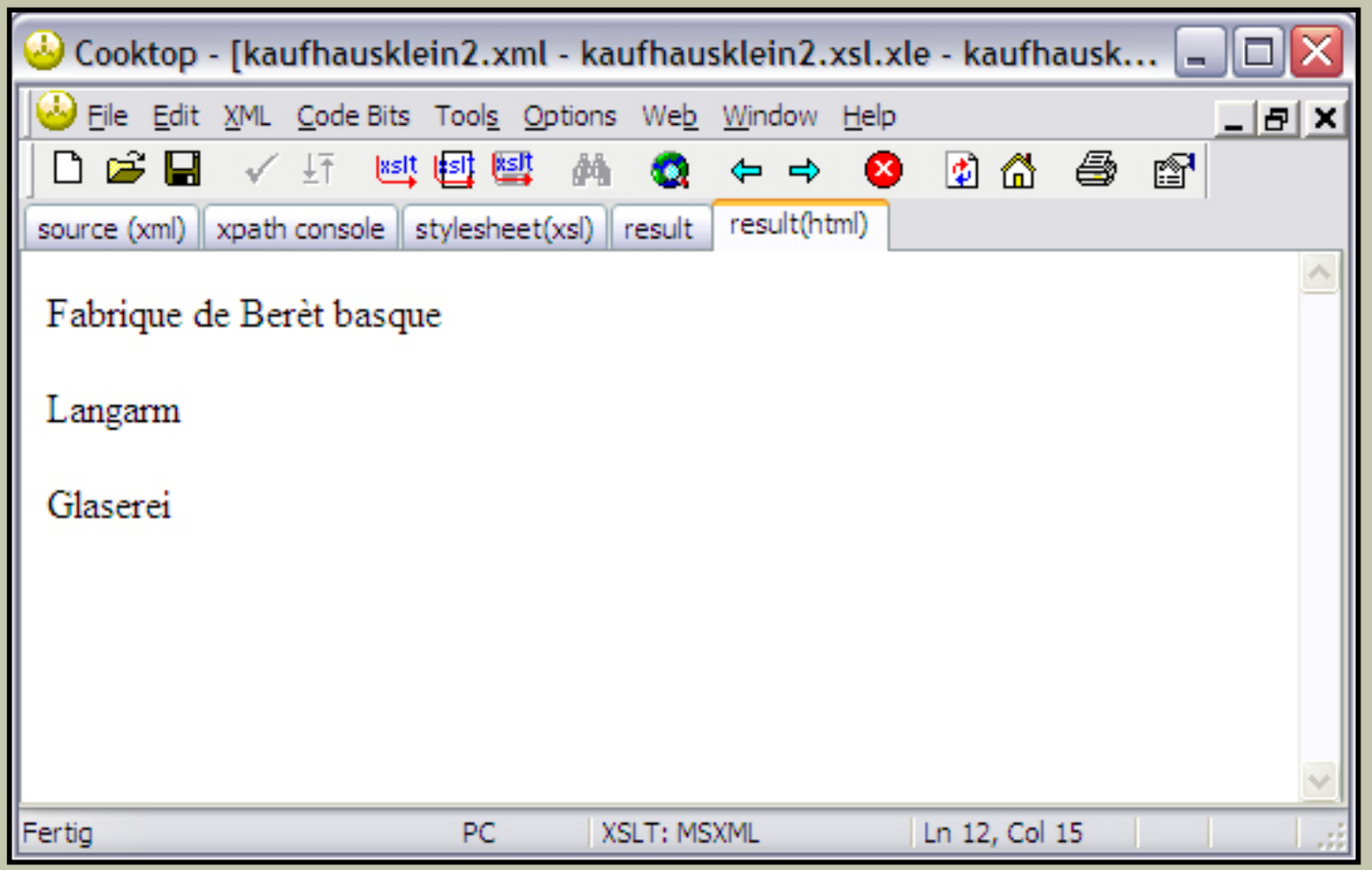
Beispiel 1.1 - Code

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="kaufhausklein2.xsl"?>
<Kaufhaus>
    <Produkt>
        <Hersteller>Fabrique de Berèt basque</Hersteller>
    </Produkt>
    <Produkt>
        <Hersteller>Langarm</Hersteller>
    </Produkt>
    <Produkt>
        <Hersteller>Glaserei</Hersteller>
    </Produkt>
</Kaufhaus>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="Hersteller">
    <html>
        <head><title>Hersteller</title></head>
        <body><p> <xsl:value-of select="."/></p></body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

match-Attribut

Beispiel 1.1 - Ausgabe



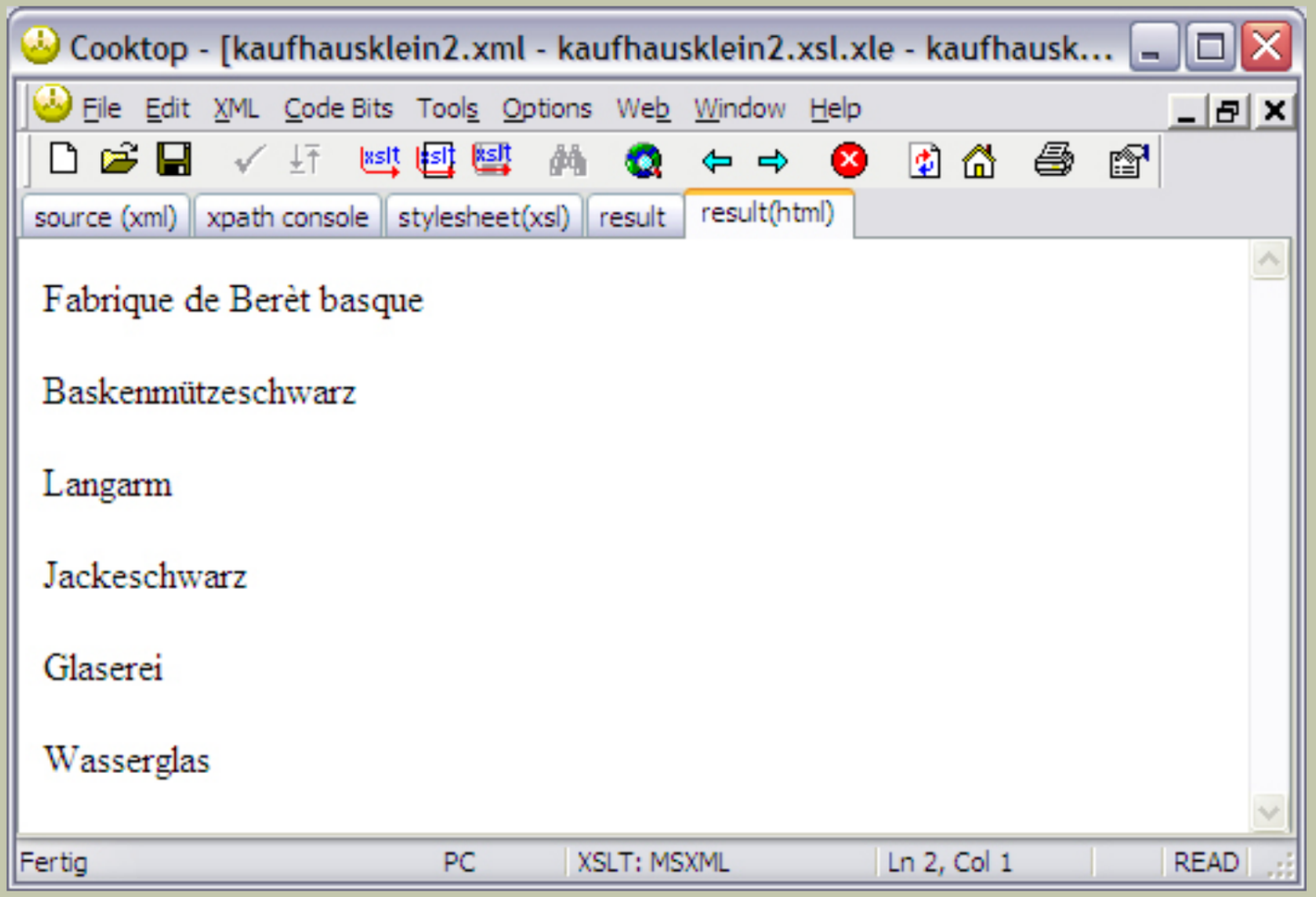
Eingebaute Template Rules

Beispiel 1.2 - Code

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="kaufhausklein2.xsl"?>
<Kaufhaus>
    <Produkt>
        <Hersteller>Fabrique de Berèt basque</Hersteller>
        <Name>Baskenmütze</Name>
        <Farbe>schwarz</Farbe>
    </Produkt>
    <Produkt>
        <Hersteller>Langarm</Hersteller>
        <Name>Jacke</Name>
        <Farbe>schwarz</Farbe>
    </Produkt>
    <Produkt>
        <Hersteller>Glaserei</Hersteller>
        <Name>Wasserglas</Name>
    </Produkt>
</Kaufhaus>
```

Eingebaute Template Rules

Beispiel 1.2 - Ausgabe



Eingebaute Template Rules

- Phänomen, dass auch ohne Templates Inhalt ausgegeben wird
- Grund: unsichtbare Template Rules
- Implizit ist im XSLT-Dokument ein (unsichtbares) Template enthalten, welches von allen Text- und Attributknoten den Inhalt ausgibt

```
<xsl:template match="text()|@*">
    <xsl:value-of select="."/>
</xsl:template>
```

- Um Ausgabe zu verhindern, muss dieses Template überschrieben werden.

  Vergleichbar mit Standardkonstruktoren in der Java-Programmierung.

```
<xsl:template match="text()|@*" />
```

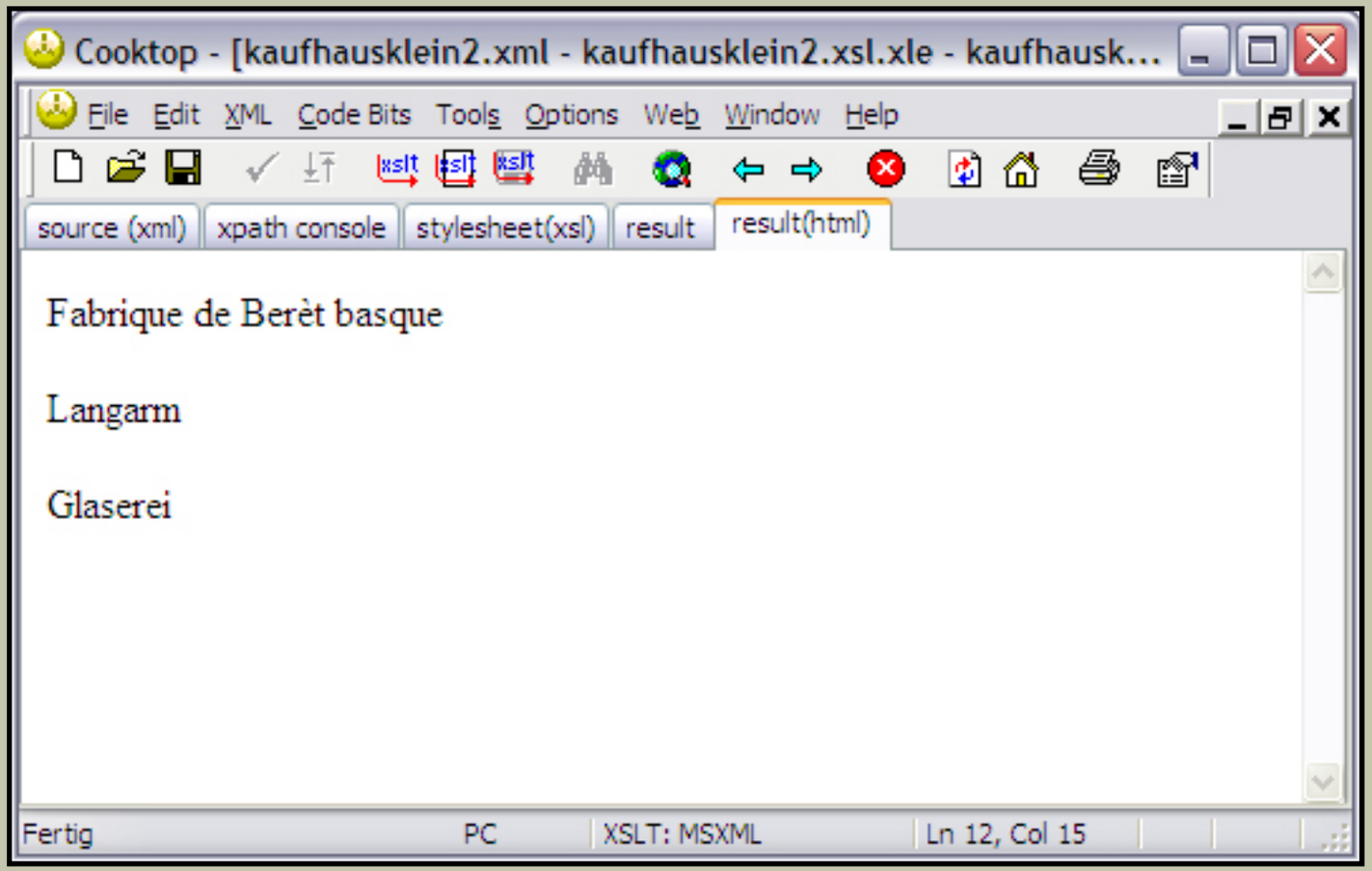
Eingebaute Template Rules

Beispiel 1.3 - Code

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="Hersteller">
    <html>
        <head><title>Hersteller</title></head>
        <body><p> <xsl:value-of select="."/></p></body>
    </html>
</xsl:template>
<xsl:template match="text()|@*" />
```

Eingebaute Template Rules

Beispiel 1.3 - Ausgabe



xsl:apply-templates

Hintergrund:

- Prozessor sucht als erstes nach passendem Template für Wurzelknoten („/“)
- arbeitet es bei Fund ab
- arbeitet danach nur weiter, wenn er innerhalb des ersten Templates die Anweisung bekommt, andere Knoten weiterzubearbeiten

- Bearbeitung aller Kindknoten

```
<xsl:apply-templates />
```

- Alternativ: Angabe eines XPath-Ausdruck

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

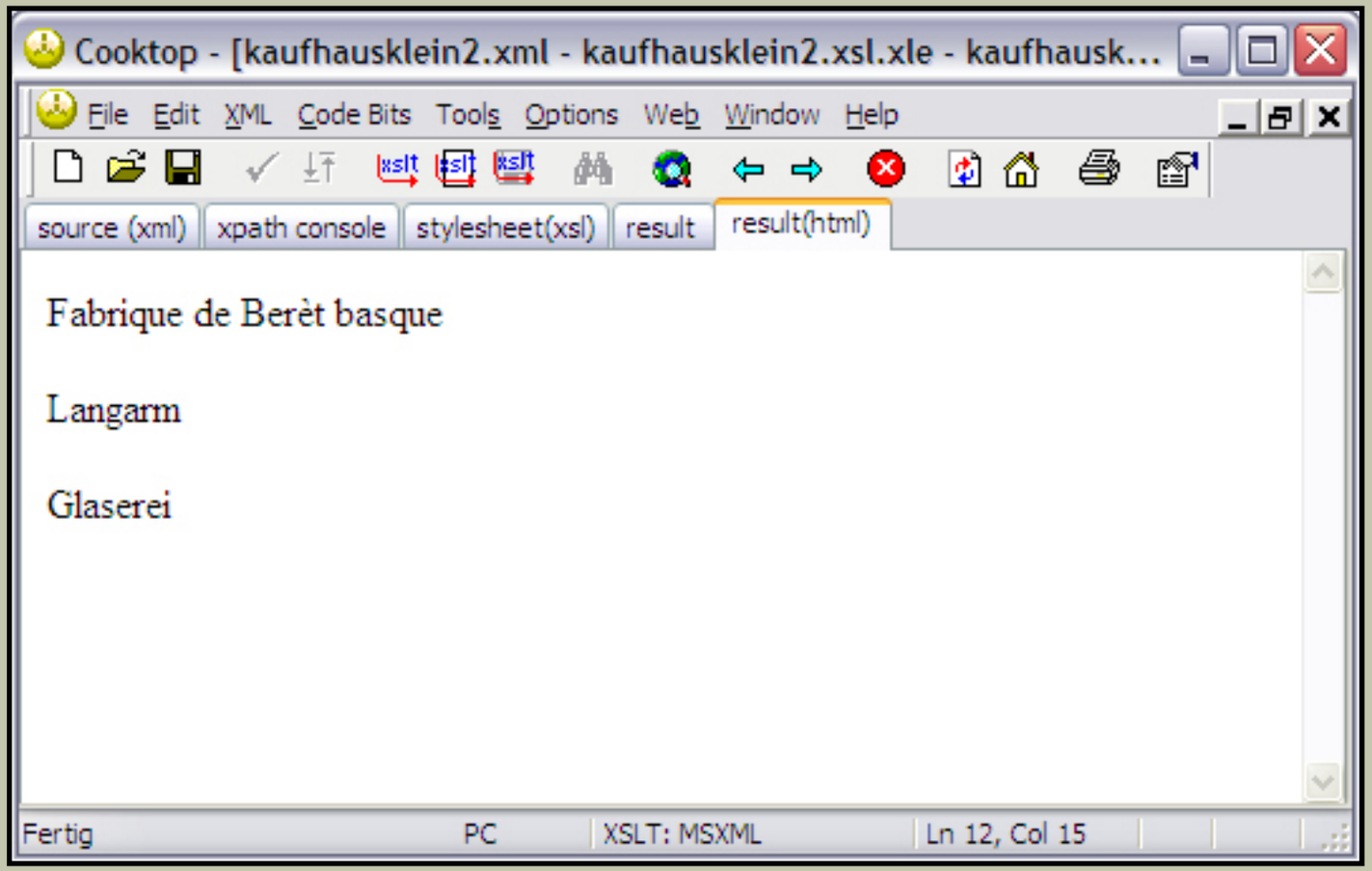
xsl:apply-templates

Beispiel 1.4 - Code

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
    <xsl:apply-templates />
</xsl:template>

<xsl:template match="Hersteller">
    <p><xsl:value-of select="."/></p>
</xsl:template>

<xsl:template match="text()|@*" />
</xsl:stylesheet>
```

xsl:apply-templates

Beispiel 1.4 - Ausgabe



xsl:apply-templates

Beispiel 1.5 - Code

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="Produkt">
    <xsl:apply-templates select="Hersteller"/>
</xsl:template>

<xsl:template match="Hersteller">
    <p><xsl:value-of select="."/></p>
</xsl:template>

</xsl:stylesheet>
```

Keine Maßnahme gegen die eingebaute Template Rule mehr nötig, da nach erstem Durchlauf der Prozessor alle Produkte behandelt hat und er aufgefordert wird, nur die „Hersteller“-Elemente zu bearbeiten, wonach es keine nicht behandelten Elemente mehr gibt.

xsl:apply-templates

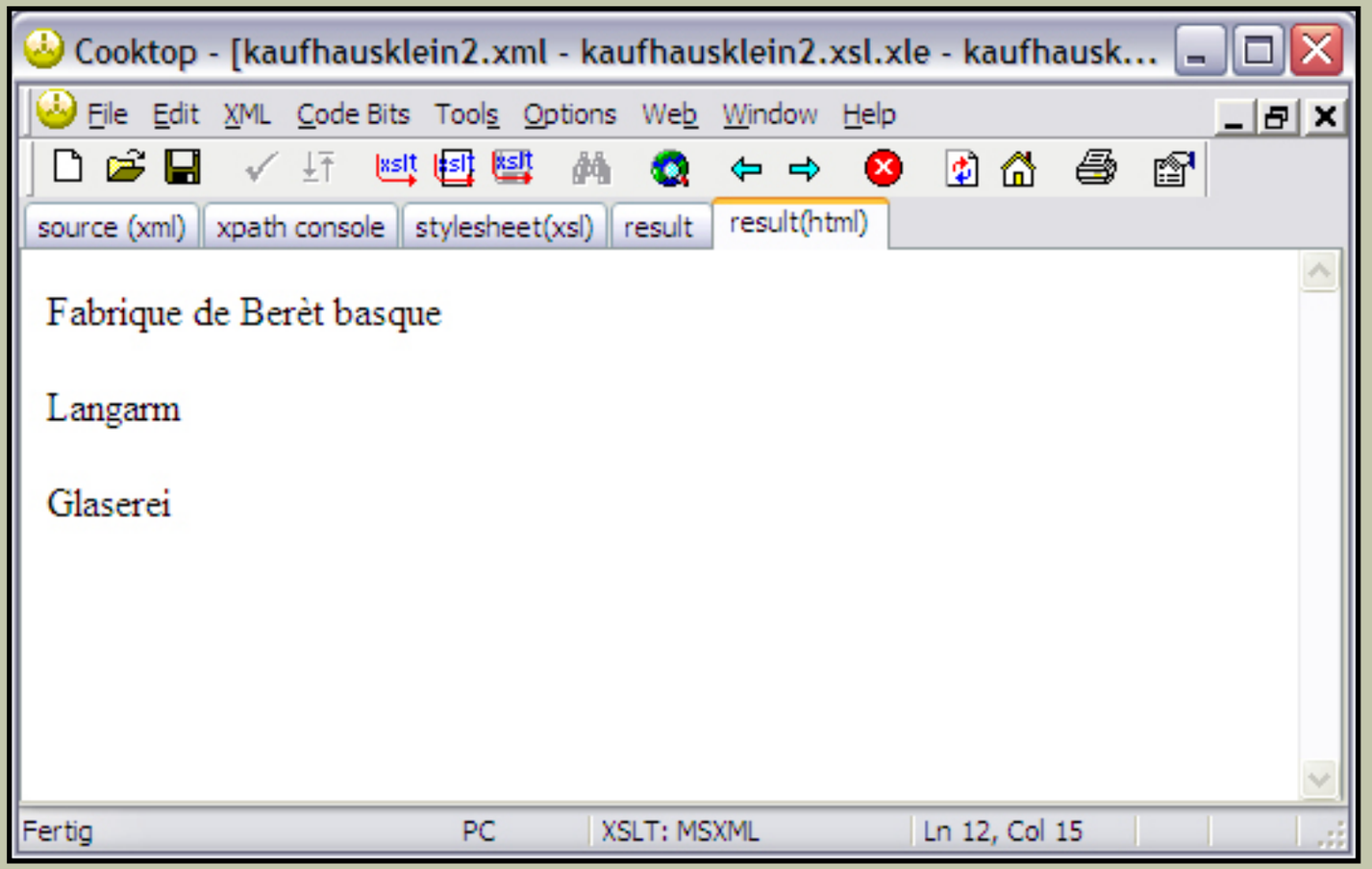
Beispiel 1.5 b - Code

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
    <xsl:apply-templates select="Kaufhaus/Produkt/Hersteller"/>
</xsl:template>

<xsl:template match="Hersteller">
    <p><xsl:value-of select="."/></p>
</xsl:template>

</xsl:stylesheet>
```

Wenn ein Zugriff auf das „Hersteller“-Element vom Wurzelknoten aus stattfinden soll, muss die Hierarchie der XML Datei aufgegriffen werden.

xsl:apply-templates

Beispiel 1.5 - Ausgabe



xsl:call-template

- Möglichkeit, Template aufzurufen, ohne dass sich der aktuelle Knoten ändert
- entsprechendes Template muss über ein name-Attribut verfügen

```
<xsl:call-template name="templateName" />
```

- Vergleichbar mit Funktionsaufruf in anderen Programmiersprachen.

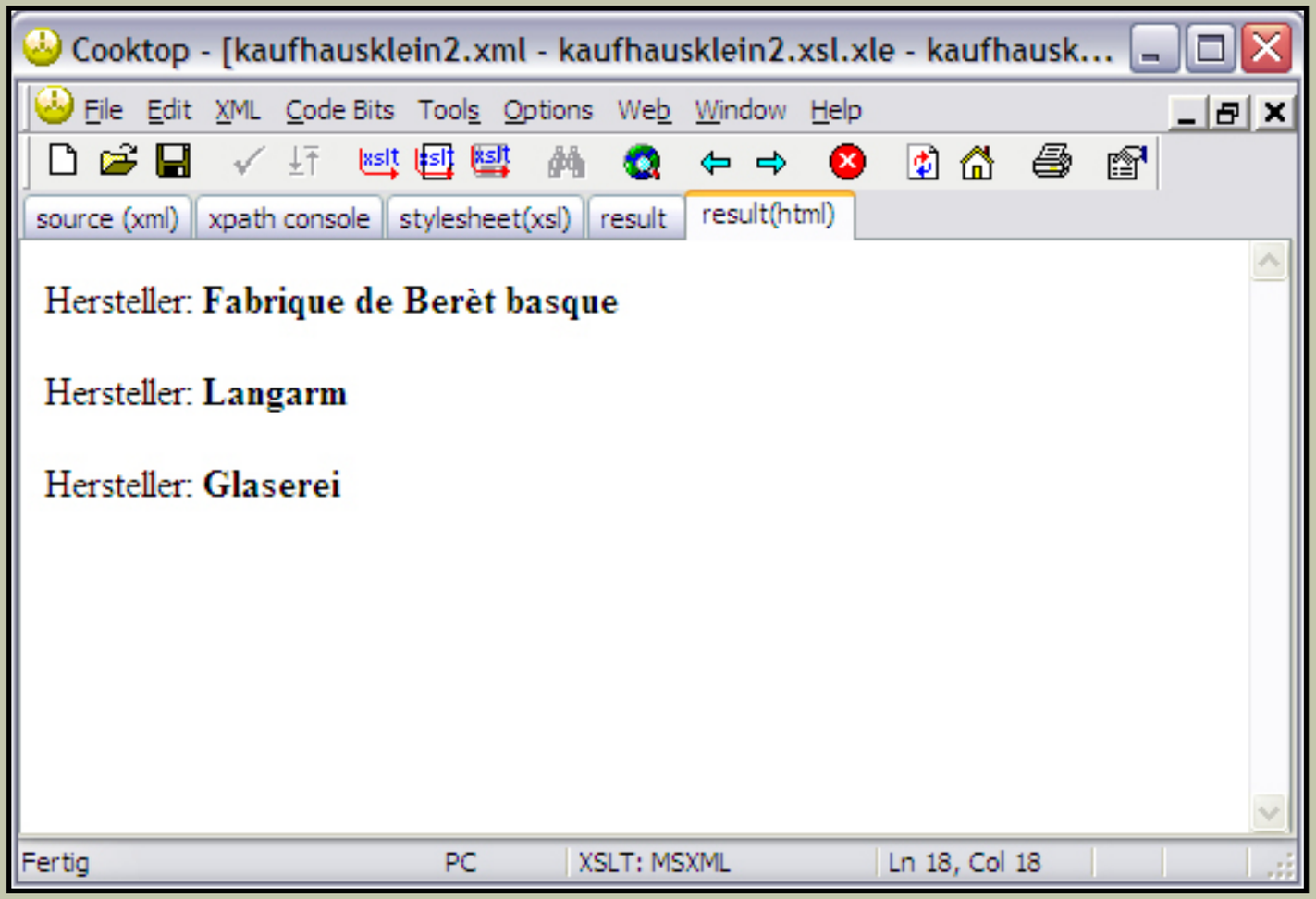
xsl:call-template

Beispiel 1.6 - Code

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <xsl:apply-templates select="Kaufhaus/Produkt/Hersteller"/>
</xsl:template>

<xsl:template name="producer">
    Hersteller:
</xsl:template>

<xsl:template match="Hersteller">
    <p>
        <xsl:call-template name="producer" />
        <b><xsl:value-of select="."/></b>
    </p>
</xsl:template>
</xsl:stylesheet>
```

xsl:call-template

Beispiel 1.6 - Ausgabe



Prioritäten zwischen Templates

- wenn mehrere Templates passend sind, wird das mit dem spezifischsten Match-Ausdruck genutzt
- Beispiel: match=“Absatz“ genauer als match=“text()“

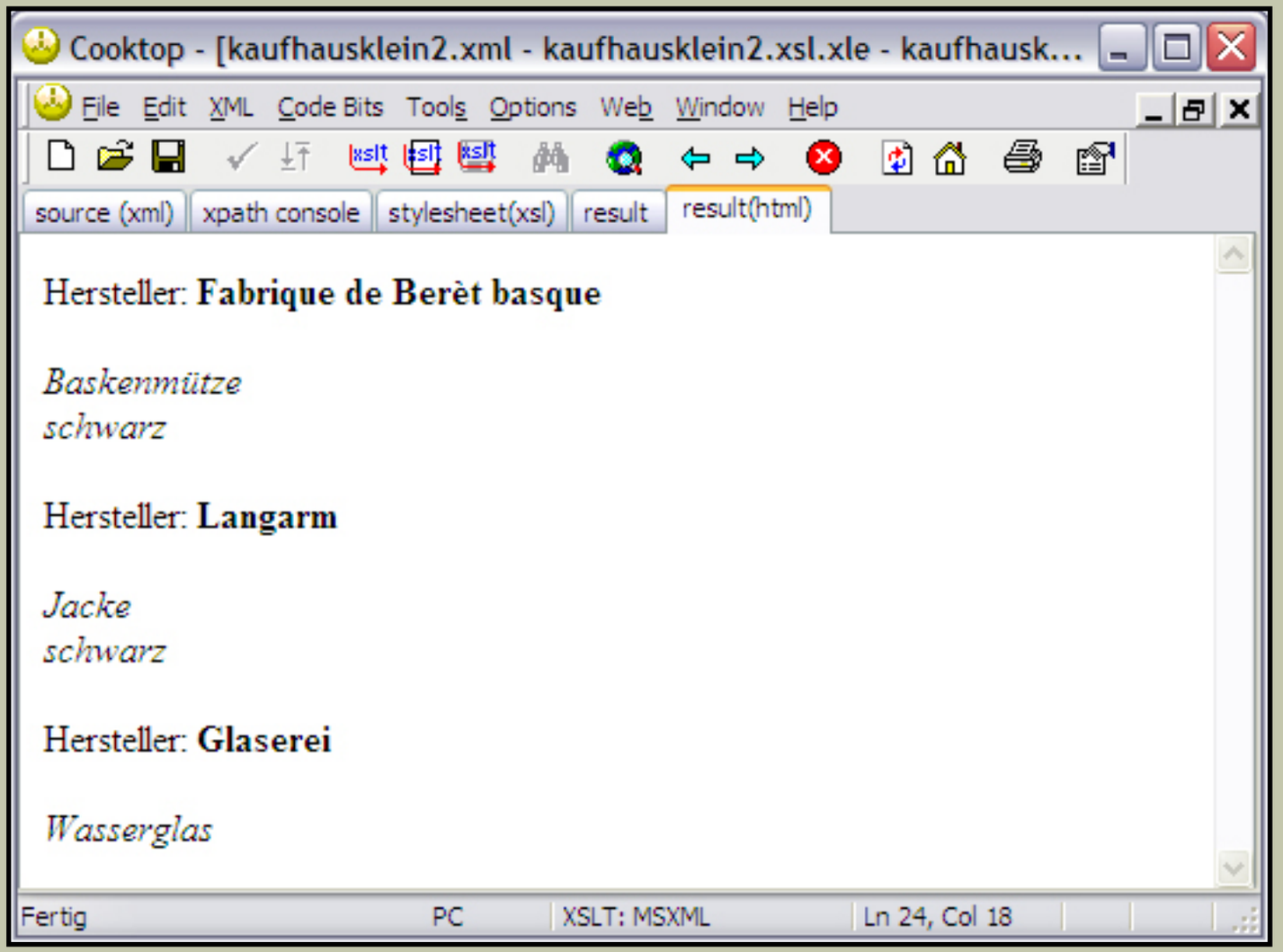
Prioritäten zwischen Templates

Beispiel 1.7 - Code

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
    <xsl:apply-templates select="Kaufhaus/Produkt"/>
</xsl:template>

<xsl:template name="producer">
    Hersteller:
</xsl:template>

<xsl:template match="text()">
    <i><xsl:value-of select="."/></i><br/>
</xsl:template>

<xsl:template match="Hersteller">
    <p>
        <xsl:call-template name="producer" />
        <b><xsl:value-of select="."/></b>
    </p>
</xsl:template>

</xsl:stylesheet>
```

Prioritäten zwischen Templates

Beispiel 1.7 - Code



Prioritäten zwischen Templates

- wenn mehrere Templates passend sind, wird das mit dem spezifischsten Match-Ausdruck genutzt
- Beispiel: match="Absatz" genauer als match="text()"

- Bei gleichem match-Attribut kann dem Template eine Priorität zugewiesen werden (Werte von -9 bis 9 möglich)

```
<xsl:template name="templateName" match="muster" priority="nummer">
</xsl:template>
```

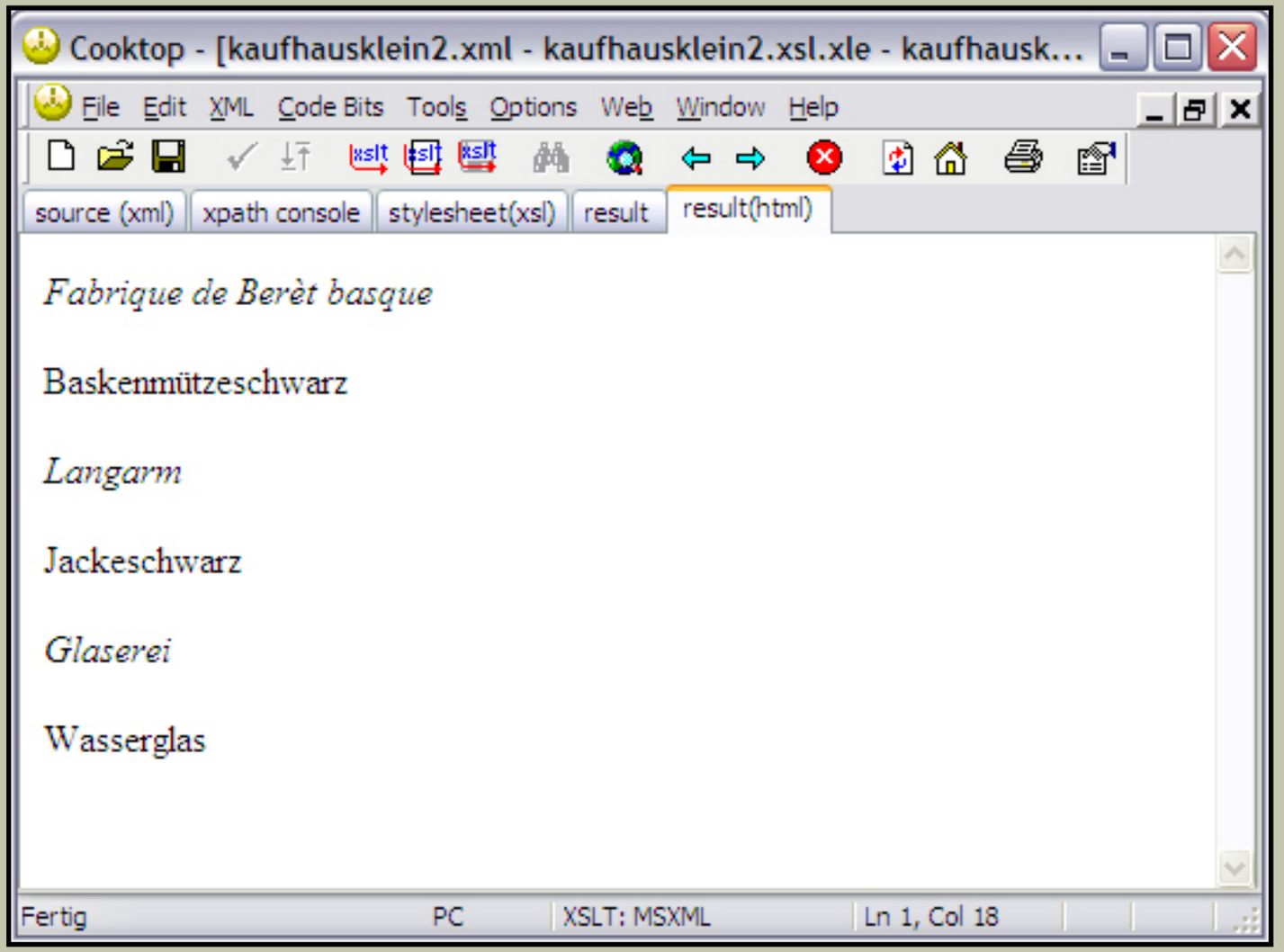
Prioritäten zwischen Templates

Beispiel 1.8 - Code

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
    <xsl:apply-templates select="Kaufhaus/Produkt"/>
</xsl:template>

<xsl:template name="producer">
    Hersteller:
</xsl:template>

<xsl:template match="Hersteller" priority="1">
    <p><i><xsl:value-of select="."/></i></p>
</xsl:template>

<xsl:template match="Hersteller" priority="-1">
    <p>
        <xsl:call-template name="producer" />
        <b><xsl:value-of select="."/></b>
    </p>
</xsl:template>

</xsl:stylesheet>
```

Prioritäten zwischen Templates

Beispiel 1.8 - Code



Parameter an Template übergeben

- ähnlich einer Funktion können einem Template Parameter übergeben werden
- Deklaration mit xsl:param

```
<xsl:param name="parameterName" select="defaultWert">
```

- select-Attribut ist optional
- aufrufende Elemente können xsl:with-param als Kindknoten enthalten und auf diese Weise einen Wert übergeben

```
<xsl:call-template name="templateName">
    <xsl:with-param name="parameterName">parameterWert</xsl:with-param>
</xsl:call-template>
```

Parameter an Template übergeben

Beispiel 1.8 - Code (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
    <xsl:apply-templates/>
</xsl:template>

<xsl:template name="producer">
    <xsl:param name="nummer" />
    <xsl:number format="{$nummer}"/>
    Hersteller:
    <xsl:apply-templates />
    <br/>
</xsl:template>

<xsl:template match="text()">
    <br/><i><xsl:value-of select="."/></i>
</xsl:template>
```

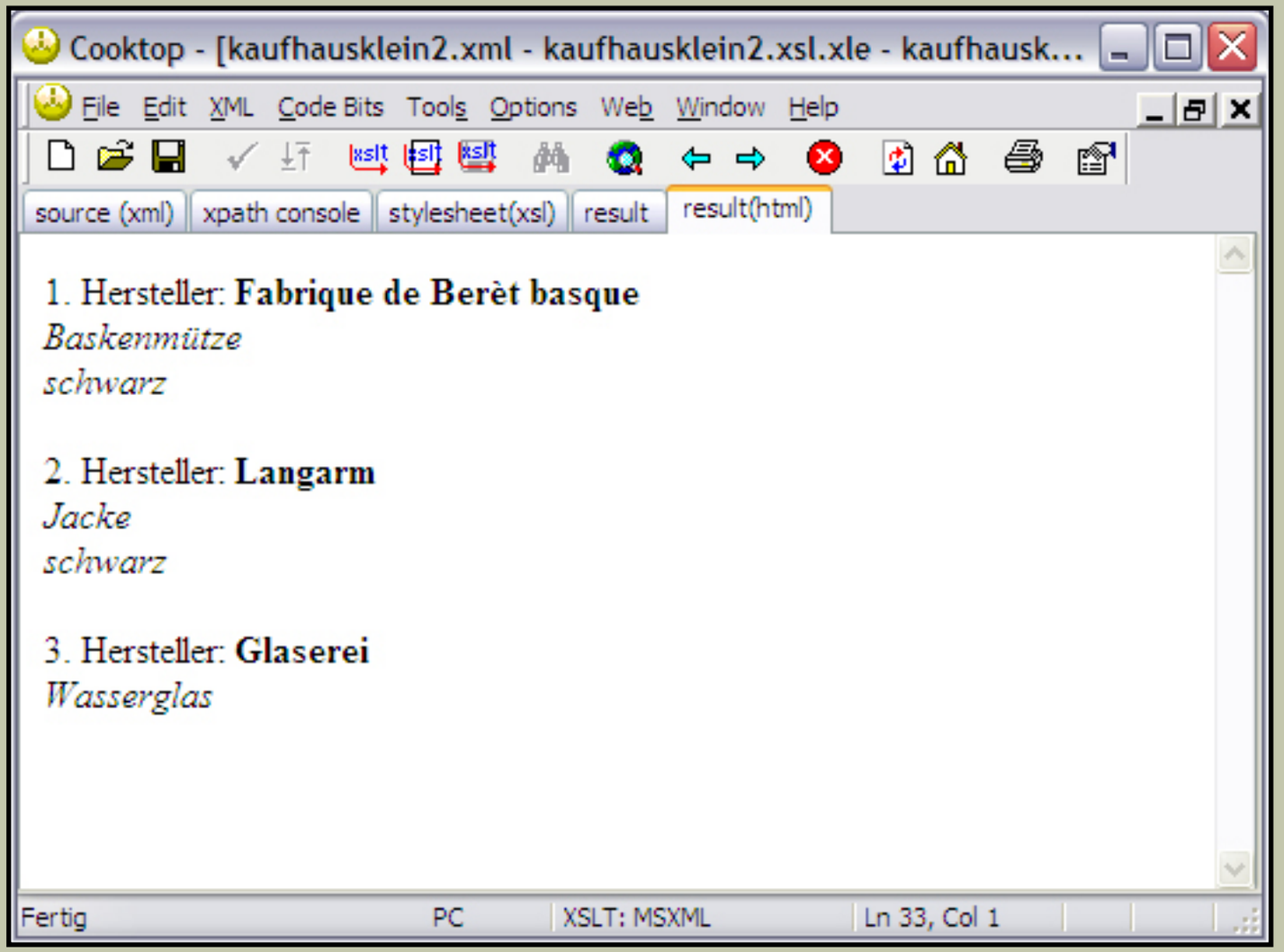
Parameter an Template übergeben

Beispiel 1.8 - Code (2)

```
<xsl:template match="Produkt">
    <p>
    <xsl:call-template name="producer">
            <xsl:with-param name="nummer"><xsl:value-of select="position()"/>.
            </xsl:with-param>
    </xsl:call-template>
    </p>
</xsl:template>

<xsl:template match="Hersteller">
        <b><xsl:value-of select="."/></b>
</xsl:template>

</xsl:stylesheet>
```

Parameter an Template übergeben

Beispiel 1.8 - Ausgabe



xsl:sort

- xsl:apply-templates bearbeitet die Knotenmenge in der Dokumentreihenfolge
- Knoten können alternativ in einer sortierten Reihenfolge bearbeitet werden

```
<xsl:apply-templates>
    <xsl:sort />
</xsl:apply-templates>
```

- mögliche Attribute

```
select = string-expression                  wonach wird sortiert
data-type = {„text“|„number“}               alphabetische oder numerische
                                            Sortierung
order = {„ascending“|„descending“}          auf- oder absteigende Sortierung
case-order = {„upper-first“|„lower-first“}  zuerst kleine oder große Buchstaben
```

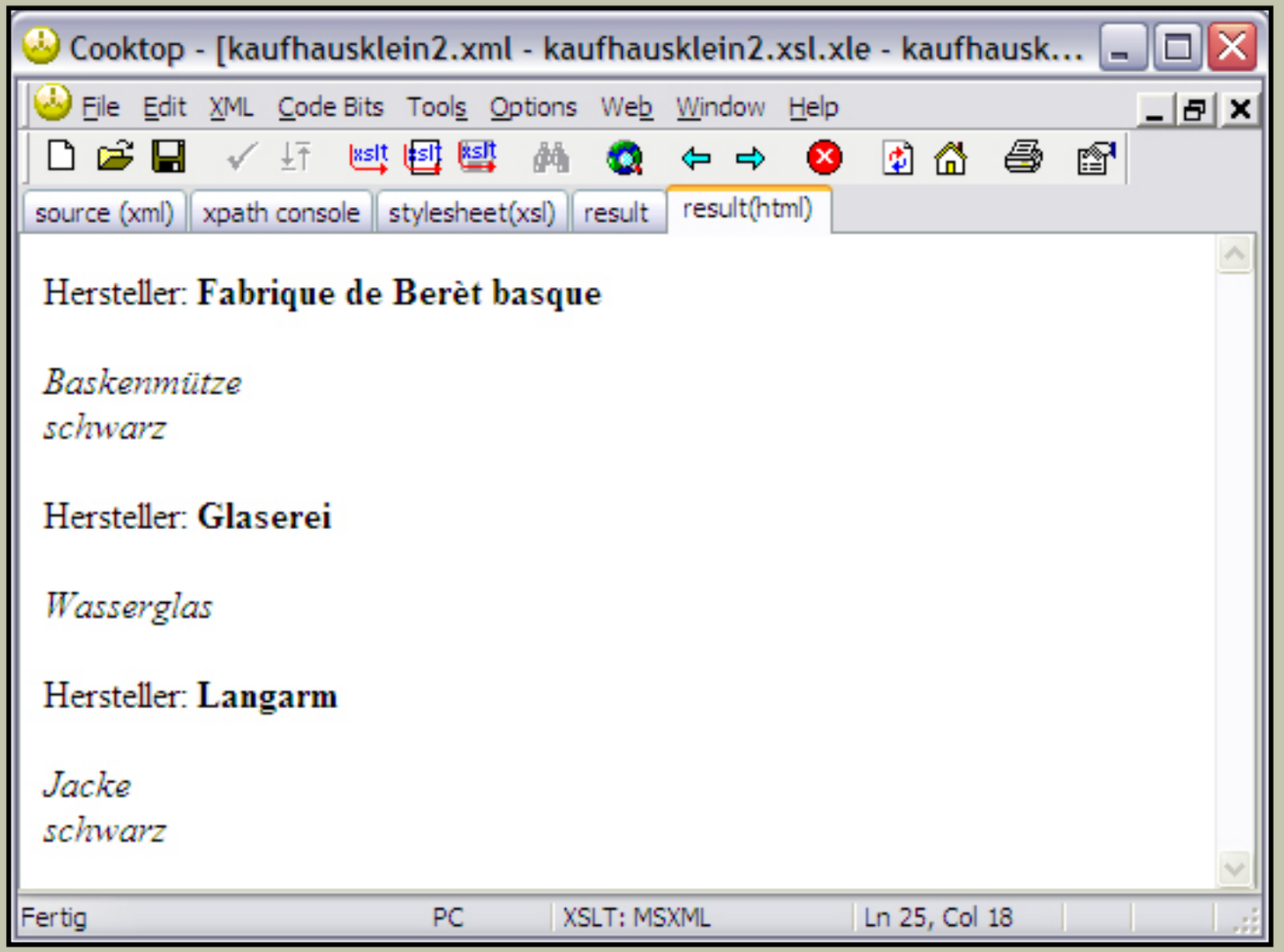
xsl:sort

Beispiel 1.9 - Code

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
    <xsl:apply-templates select="Kaufhaus/Produkt">
        <xsl:sort select="Hersteller"/>
    </xsl:apply-templates>
</xsl:template>

<xsl:template name="producer">
    Hersteller:
</xsl:template>

<xsl:template match="text()">
    <i><xsl:value-of select="."/></i><br/>
</xsl:template>

<xsl:template match="Hersteller">
    <p>
        <xsl:call-template name="producer"/>
        <b><xsl:value-of select="."/></b>
    </p>
</xsl:template>
</xsl:stylesheet>
```

xsl:sort

Beispiel 1.9 - Ausgabe



- Besondere Elemente -

xsl:text

- erzeugt Textknoten
- explizite Kontrolle über Sonderzeichen („<“ und „>“)

```
<xsl:text disable-output-escaping = „yes“></xsl:text>
```

- normalerweise werden alle Sonderzeichen umgewandelt

XSLT	Ausgabe bei disable-output-escaping = „yes“	Ausgabe bei disable-output-escaping = „no“
>	>	>
<	<	<

- default-Wert für disable-output-escaping ist „no“

xsl:output

- Beeinflussung der Datenausgabe
  z.B. Angabe einer Ausgabemethode und Encoding des Zieldokuments

```
<xsl:output
    method = „xml“ | „html“ | „text“
    encoding = string
</xsl:output>
```

- Weitere Attribute für die Ausgabemethode „xml“

Attribut	Wirkung
version = nmtoken	Deklaration der Version, derzeit nur „1.0“ möglich
omit-xml-declaration=“yes“|“no“	bei „yes“ entfällt die XML-Deklaration
indent=“yes“|“no“	bei „yes“ Umbrüche wie im XSLT

- Ausgabemethode „text“ gibt keinerlei XML-Tags aus,
  sondern nur die Inhalte von Textknoten

xsl:output

Ausgabemethode „html“

- wird automatisch gewählt, wenn das erste Element der Ausgabe den Namen „html“ trägt
- XSLT Prozessor nimmt einige Modifikationen an der Ausgabe vor: leere Elemente werden so formuliert, dass sie keinen End-Tag haben Bsp.:
</br> oder
 wird zu
- XSLT Prozessor erzeugt in der HTML Ausgabe entsprechenden Meta-Tag

- Weitere Attribute der Ausgabemethode „html“

Attribut	Wirkung
media-type	zum Beispiel „text/html“ für Meta-Tag (content=“text/html; ...“)

- Problem: Dokument ist nach Umformung nicht mehr XML-kompatibel

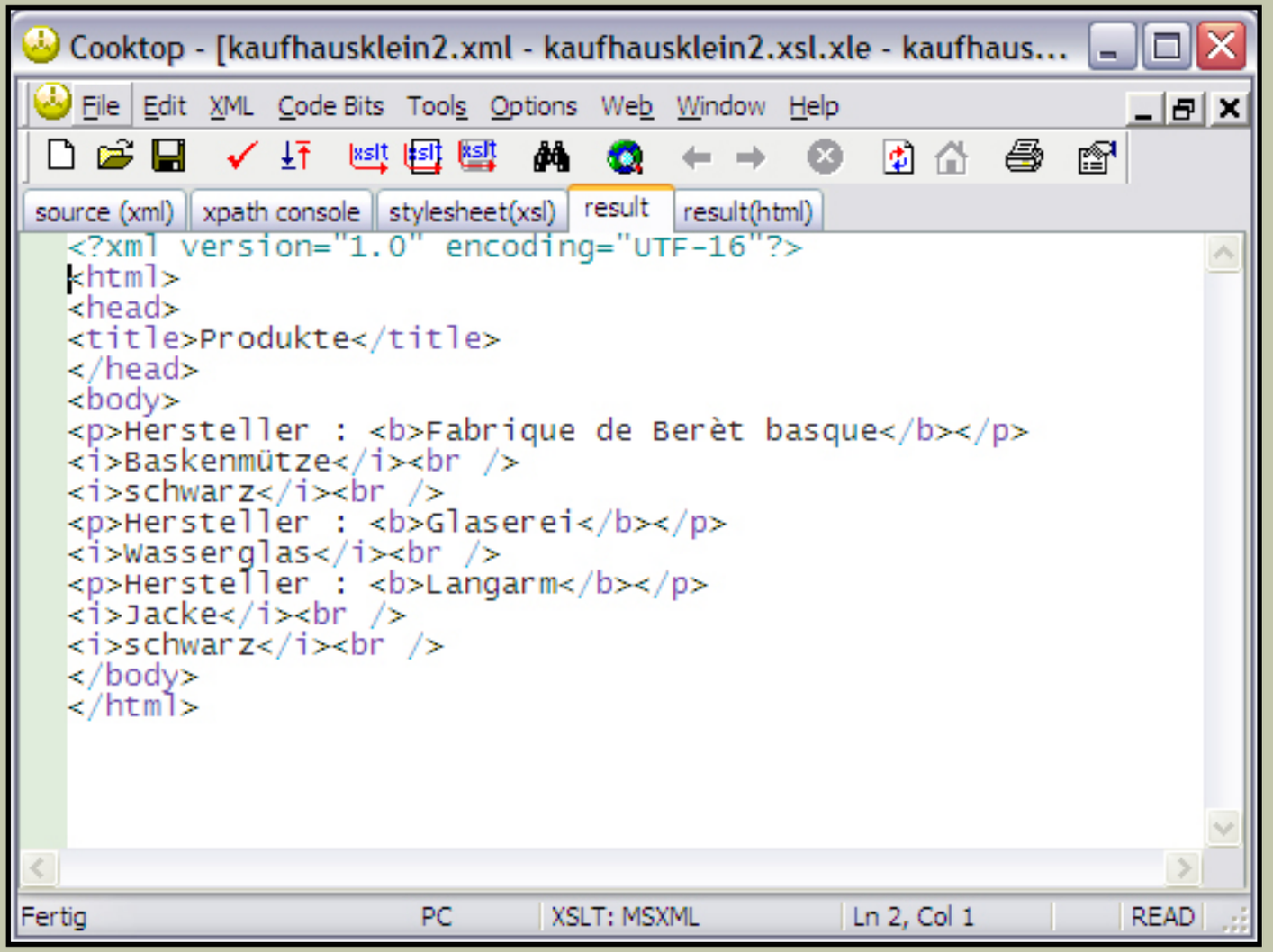
xsl:output

Beispiel 2.1 - Code (Auszug)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output
    method = „xml“
    encoding = „UTF-16“
    indent = „yes“
/>

<xsl:template match="/">
    <html>
    <head>
    <title>Produkte</title>
    </head>
    <body>
    <xsl:apply-templates select="Kaufhaus/Produkt">
        <xsl:sort select="Hersteller"/>
    </xsl:apply-templates>
    </body>
    </html>
</xsl:template>
...
```

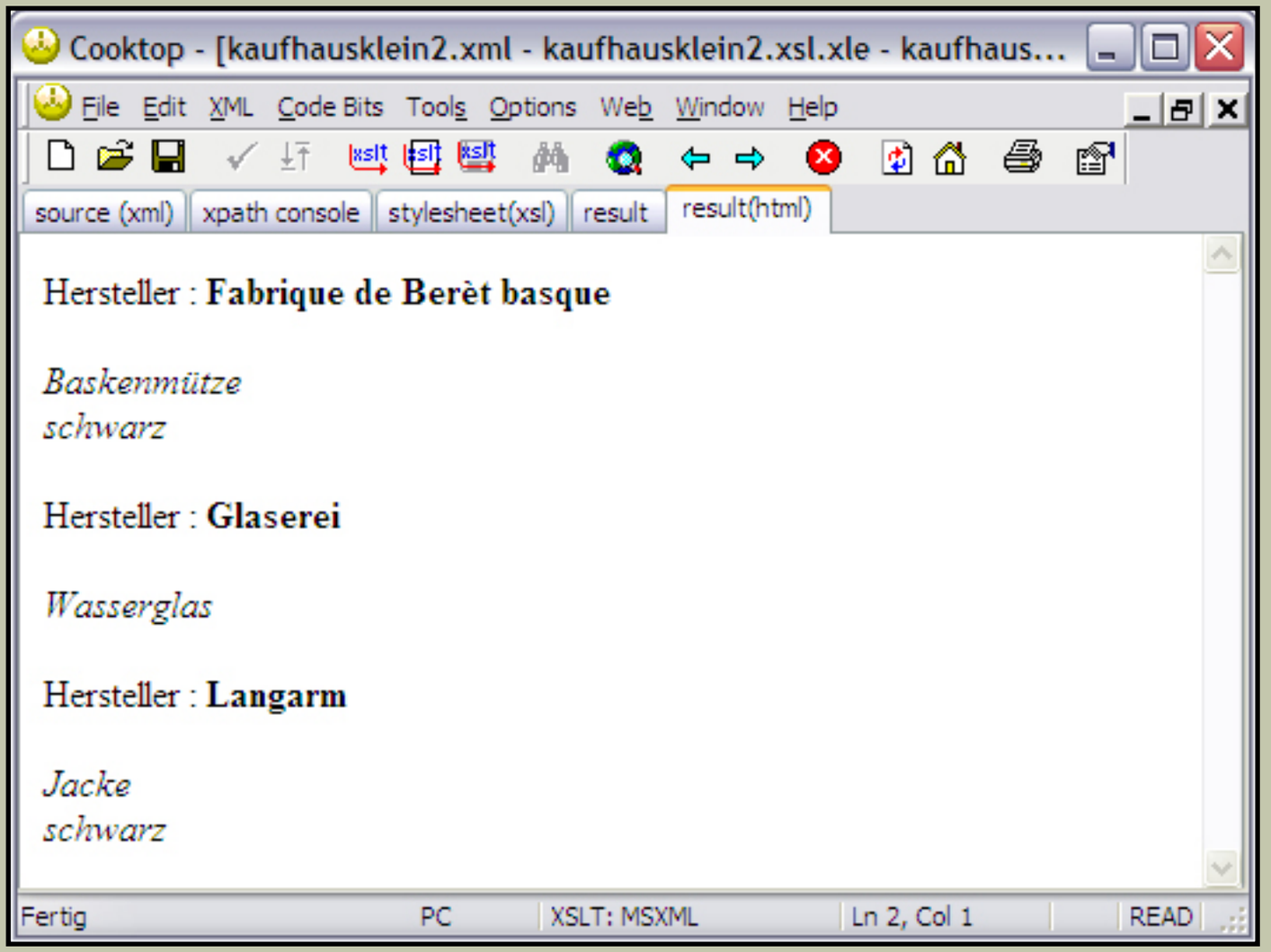
xsl:output

Beispiel 2.1 - Ausgabe (Code)



xsl:output

Beispiel 2.1 - Ausgabe (unverändert)



xsl:comment

- XML und HTML verwenden die gleiche Kommentarform:
  <!-- Kommentar -->
- Erzeugung mittels xsl:comment

```
<xsl:comment>Kommentar</xsl:comment>
```

- Kommentare für das XSLT-Dokument werden nicht mit umgeformt.

xsl:comment

Beispiel 2.2 - Code (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- Definition der Output Methode -->
<xsl:output
    method = „xml"
    encoding = „UTF-16"
    indent = „yes"
/>

<!-- Dieses Template wird als erstes aufgerufen.
     Es ist für das Wurzelelement und wird daher nur einmal ausgeführt. -->
<xsl:template match="/">
    <html>
    <head>
    <title>Produkte</title>
    </head>
    <body>
    <xsl:comment>Liste der Hersteller und ihrer Produkte</xsl:comment>
    <xsl:apply-templates select="Kaufhaus/Produkt">
        <xsl:sort select="Hersteller"/>
    </xsl:apply-templates>
    </body>
...
```

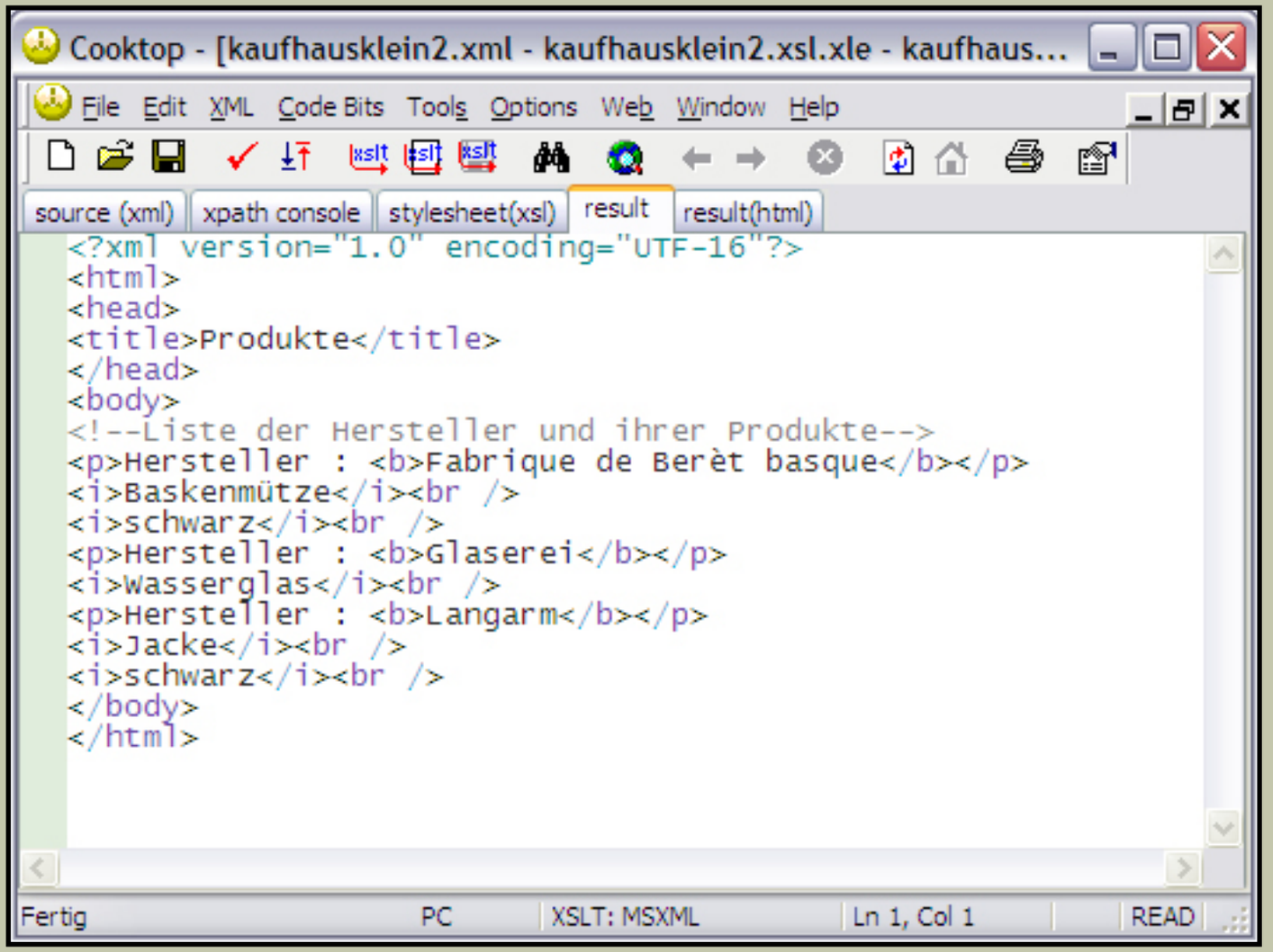
xsl:comment

Beispiel 2.2 - Code (2)

```
...
    </html>
</xsl:template>

<!-- Dieses Template wird nicht für ein konkretes Element genutzt,
     sondern muss explizit aufgerufen werden. -->
<xsl:template name="producer">
    <xsl:text>Hersteller : </xsl:text>
</xsl:template>

<!-- Dieses Template behandelt alle Textknoten -->
<xsl:template match="text()">
    <i><xsl:value-of select="."/></i><br/>
</xsl:template>

<!-- Dieses Template ist für das Element „Hersteller“ -->
<xsl:template match="Hersteller">
    <p><xsl:call-template name="producer"/>
        <b><xsl:value-of select="."/></b></p>
</xsl:template>

</xsl:stylesheet>
```

xsl:comment

Beispiel 2.2 - Ausgabe (Code)



xsl:number

- zur Feststellung der Position des aktuellen Knotens in der Quelldatei
- Möglichkeit zur Formatierung der Zahl

```
<xsl:number />
```

- mögliche Attribute und ihre Benutzung

Attribut	Wirkung
count="ausdruck"	xPath-Ausdruck der festlegt, welche Elemente gezählt werden
level="single"|"multiple"|"any"	single - nur das angegebene Element multiple - hierarchisch any - alle ungeachtet der ebene default: single
value="ausdruck"	benutzerdefinierter Wert
format="formatstring"	möglich: 1 (arabische Zahle I (römische Zahlen, groß) i (römische Zahlen, klein) a (kleine Buchstaben) A (große Buchstaben)

xsl:number

- mögliche Attribute und ihre Benutzung (Fortsetzung)

Attribut	Wirkung
grouping-separator="character"	Trennzeichen default: „," (z.B. 1,000)
grouping-size="number"	Gruppierung von Ziffern default: „3" (100,000)

- weitere Attribute:

```
from="ausdruck"
lang="languagecode"
letter-value="alphabetic"|"traditional"
```

- für außereuropäische Sprachen kann eine weitergehende Kombination der Attribute „lang" und „letter-value" nötig sein

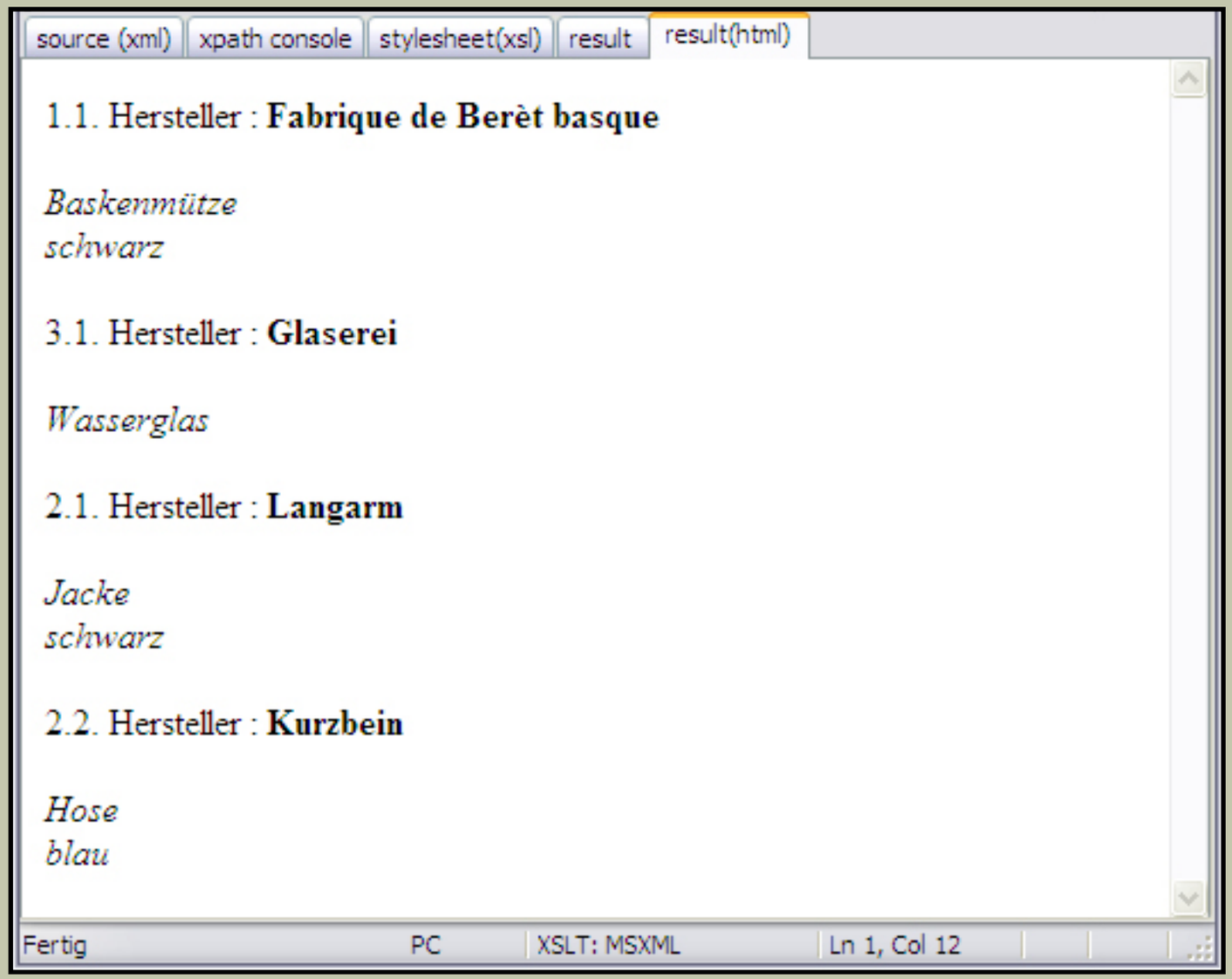
xsl:number

Beispiel 2.3 - Code (Auszug)

```
...
<!-- Dieses Template wird nicht für ein konkretes Element genutzt,
     sondern muss explizit aufgerufen werden. -->
<xsl:template name="producer">
    <xsl:text>Hersteller : </xsl:text>
</xsl:template>

<!-- Dieses Template behandelt alle Textknoten -->
<xsl:template match="text()">
    <i><xsl:value-of select="."/></i><br/>
</xsl:template>

<!-- Dieses Template ist für das Element „Hersteller" -->
<xsl:template match="Hersteller">
    <p><xsl:number level="multiple" count="Produkt|Hersteller" format="1.1. "/>
        <xsl:call-template name="producer"/>
        <b><xsl:value-of select="."/></b></p>
</xsl:template>

</xsl:stylesheet>
```

xsl:number

Beispiel 2.3 - Ausgabe (zusätzliches Element in XML)



xsl:variable

- XSLT bietet keine Variablen im herkömmlichen Sinne an
- xsl:variable kann nur einmalig initialisiert und nicht nachträglich verändert werden
- xsl:variable entspricht in seiner Definition einer Konstante

```
<xsl:variable name="ort">Greifswald</xsl:variable>
```

- Referenzierung durch vorangestelltes Dollarzeichen

```
Sommerakademie in <xsl:value-of select="$ort"/>
```

- zwei Varianten der Initialisierung

```
<!-- Das erste item-Element wird ausgegeben -->
<xsl:variable name="n">2</xsl:variable>
<xsl:value-of select="item[$n]" />

<!-- Das zweite item-Element wird ausgegeben -->
<xsl:variable name="n" select="2" />
<xsl:value-of select="item[$n]" />
```

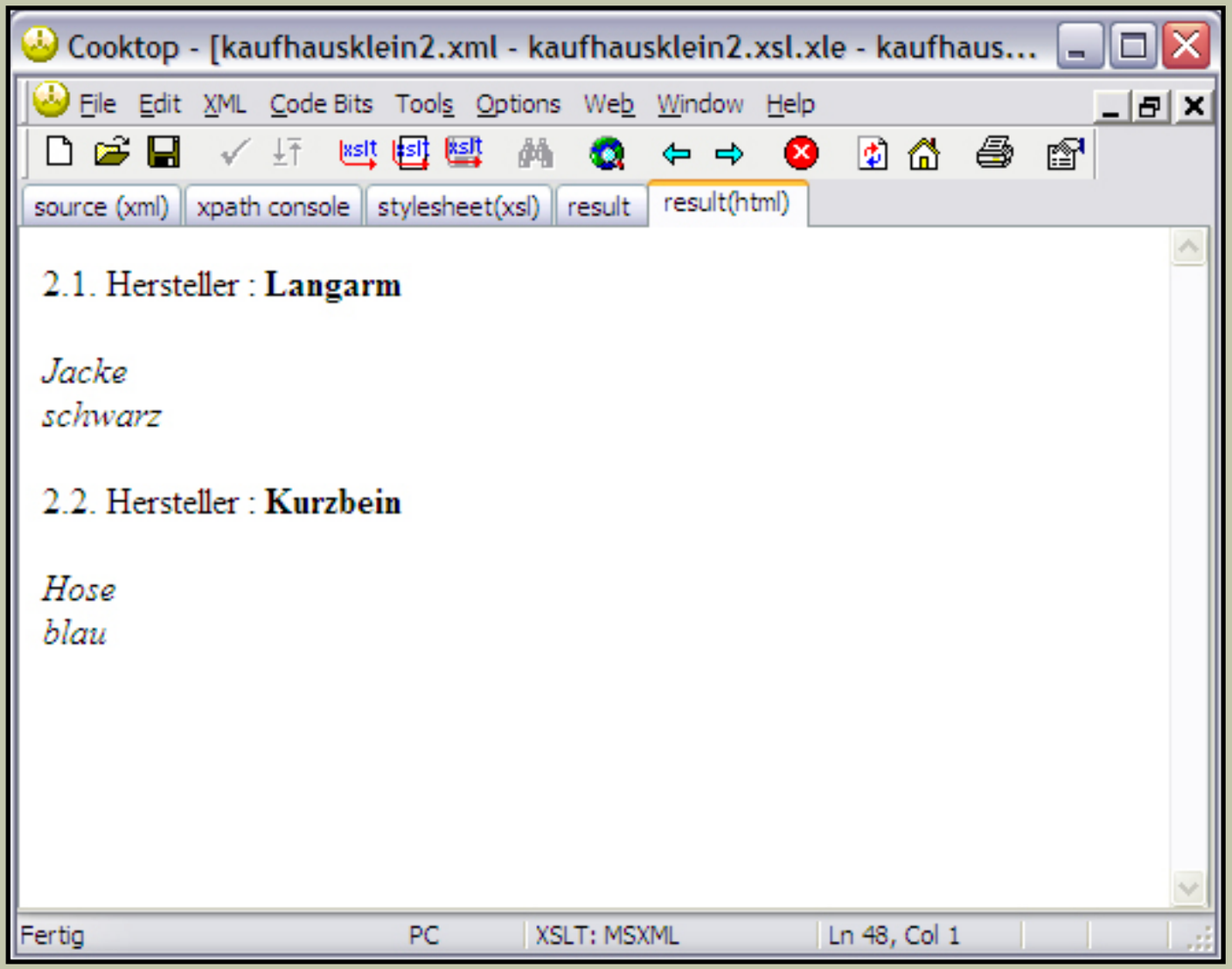
xsl:variable

Beispiel 2.4 - Code (Auszug)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- Definition der Output Methode -->
<xsl:output
    method = „xml"
    encoding = „UTF-16"
    indent = „yes"
/>

<xsl:variable name="ausgabeNummer" select="2" />

<!-- Dieses Template wird als erstes aufgerufen.
     Es ist für das Wurzelelement und wird daher nur einmal ausgeführt. -->
<xsl:template match="/">
    <html>
    <head>
    <title>Produkte</title>
    </head>
    <body>
    <xsl:comment>Liste der Hersteller und ihrer Produkte</xsl:comment>
    <xsl:apply-templates select="Kaufhaus/Produkt[$ausgabeNummer]">
        <xsl:sort select="Hersteller"/>
    </xsl:apply-templates>
...
```

xsl:variable

Beispiel 2.4 - Ausgabe



- Kontrollstrukturen -

Einfache Bedingung

- für einfach Bedingungen xsl:if
- es existiert für xsl:if kein else, wie in vielen anderen Programmiersprachen

```
<xsl:if test="boolscher Ausdruck"></xsl:if>
```

- das test-Attribut muss angegeben werden
- der Ausdruck des test-Attributs wird in einen logischen Wert umgewandelt
- gibt der Ausdruck eine Knotenmenge zurück, gilt der Wert als „wahr“

Interessante XPath-Knotensatzfunktionen

```
number last()                    Anzahl der Knoten im aktuellen Knotensatz
number position()                Position des aktuellen Knotens im Knotensatz
number count(node-set)           Anzahl der Knoten im übergebenen Knotensatz
number name(node-set)            Name des ersten Knotens
```

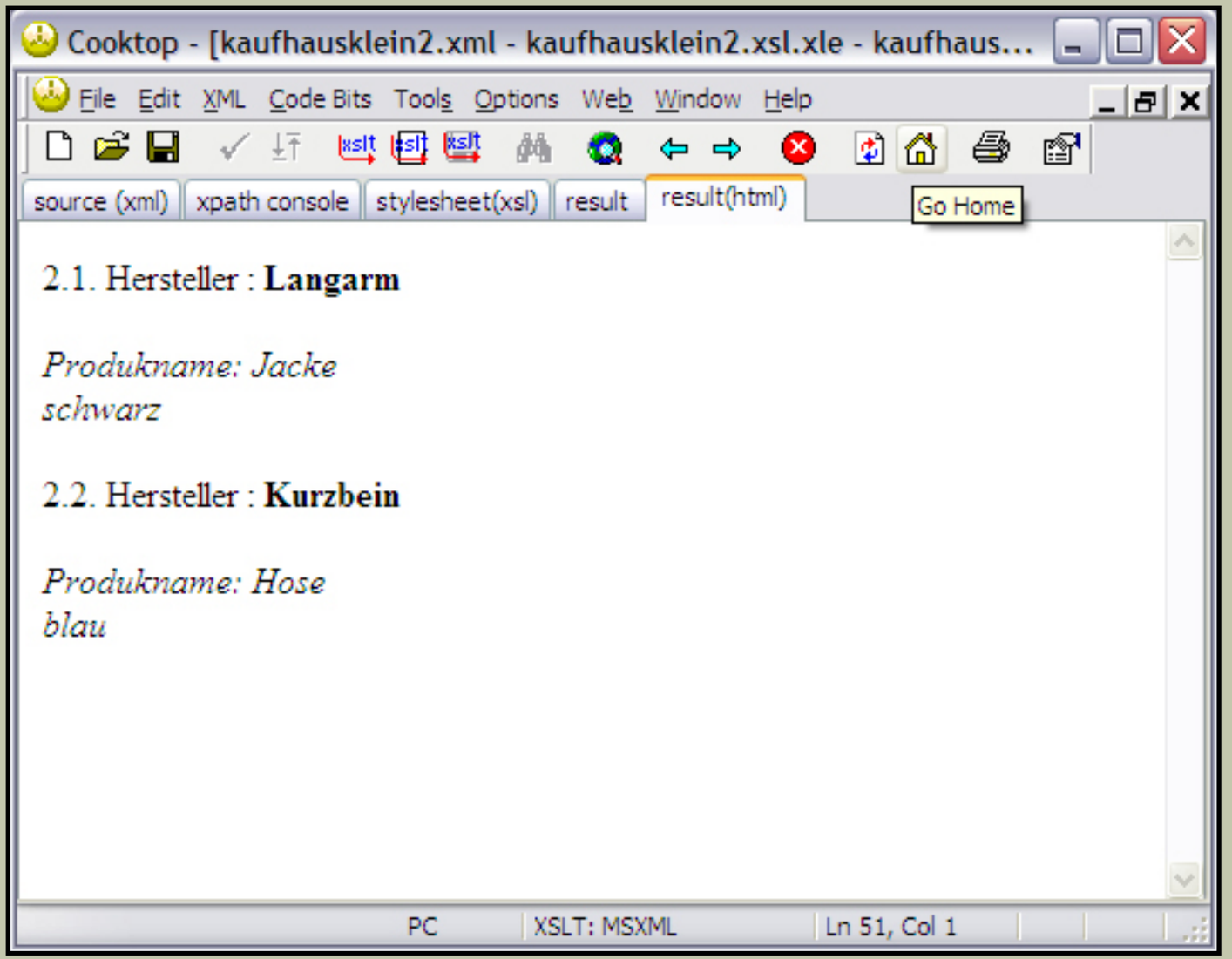
Einfache Bedingung

Beispiel 3.1 - Code (Auszug)

```
...

<!-- Dieses Template wird nicht für ein konkretes Element genutzt,
     sondern muss explizit aufgerufen werden. -->
<xsl:template name="producer">
    <xsl:text>Hersteller : </xsl:text>
</xsl:template>

<!-- Dieses Template behandelt alle Textknoten -->
<xsl:template match="text()">
    <xsl:if test="name(..)='Name'">
            <i>Produkname: </i>
    </xsl:if>
    <i><xsl:value-of select="."/></i><br/>
</xsl:template>

<!-- Dieses Template ist für das Element „Hersteller" -->
<xsl:template match="Hersteller">
    <p><xsl:number level="multiple" count="Produkt|Hersteller" format="1.1. „/>
        <xsl:call-template name="producer"/>
        <b><xsl:value-of select="."/></b></p>
</xsl:template>

</xsl:stylesheet>
```

Einfache Bedingung

Beispiel 3.1 - Ausgabe



Mehrere Optionen

- Möglichkeit eines „else“-Zweiges mit xsl:when und xsl:otherwise
- xsl:when und xsl:otherwise sind Kindelemente von xsl:choose

```
<xsl:choose>
    <xsl:when test="boolscher Ausdruck">
    </xsl:when>
    <xsl:otherwise>
    </xsl:otherwise>
</xsl:choose>
```

- eine xsl:choose-Klammer kann beliebig viele xsl:when-Elemente und genau ein (oder kein) xsl:otherwise-Element enthalten
- trifft eines der xsl:when-Elemente zu, wird dessen Inhalt ausgeführt und die xsl:choose-Klammer verlassen
  (Eine break-Anweisung wie etwa bei der Java-Programmierung ist hier nicht nötig)

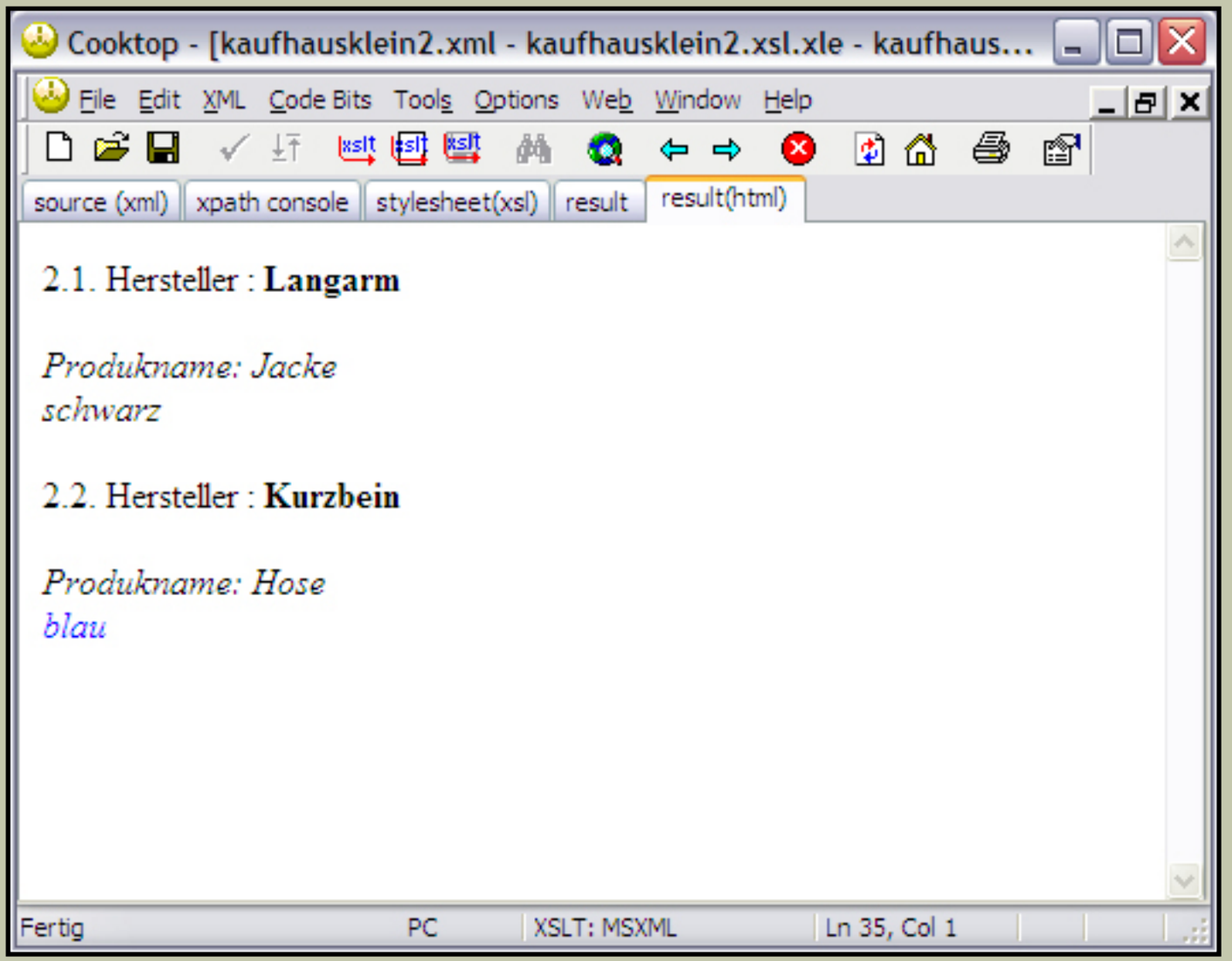
Mehrere Optionen

Beispiel 3.2 - Code (Auszug)

```
...
<!-- Dieses Template behandelt alle Textknoten -->
<xsl:template match="text()">
    <xsl:if test="name(..)='Farbe'">
        <xsl:variable name="farbe"><xsl:value-of select="."/></xsl:variable>
        <xsl:choose>
            <xsl:when test="$farbe='blau'">
                <i style="color: #0000FF;"><xsl:value-of select="."/></i>
            </xsl:when>
            <xsl:otherwise>
                <i><xsl:value-of select="."/></i>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:if>
    <xsl:if test="name(..)='Name'">
            <i>Produkname: <xsl:value-of select="."/></i><br/>
    </xsl:if>
</xsl:template>
...
```

Mehrere Optionen

Beispiel 3.2 - Ausgabe



Schleifen

- xsl:for-each erlaubt Bearbeitung einer Serie von Knoten
- Besonderheit: Sobald ein Knoten bearbeitet wird, wird er zum aktuellen Knoten.
- select-Attribut bestimmt Knoten

```
<xsl:for-each select="ausdruck">
</xsl:for-each>
```

- current() bezeichnet das aktuelle Element
- "." als Abkürzung für current() möglich

Schleifen

Beispiel 3.3 - Code (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="kaufhausklein2.xsl"?>
<Kaufhaus>
    <Produkt>
        <Hersteller>Fabrique de Berèt basque</Hersteller>
        <Name>Baskenmütze</Name>
        <Farbe>rot</Farbe>
    </Produkt>
    <Produkt>
        <Hersteller>Langarm</Hersteller>
        <Name>Jacke</Name>
        <Farbe>schwarz</Farbe>
        <Hersteller>Kurzbein</Hersteller>
        <Name>Hose</Name>
        <Farbe>blau</Farbe>
    </Produkt>
    <Produkt>
        <Hersteller>Glaserei</Hersteller>
        <Name>Wasserglas</Name>
    </Produkt>
</Kaufhaus>
```

Schleifen

Beispiel 3.3 - Code (XSLT) (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- Definition der Output Methode -->
<xsl:output
    method = „xml"
    encoding = „UTF-16"
    indent = „yes"
/>

<!-- Dieses Template wird als erstes aufgerufen.
     Es ist für das Wurzelelement und wird daher nur einmal ausgeführt. -->
<xsl:template match="/">
    <html>
    <head>
    <title>Produkte</title>
    </head>
    <body>
    <xsl:comment>Liste der Hersteller und ihrer Produkte</xsl:comment>

    <xsl:for-each select="Kaufhaus/Produkt">
        <!-- Diese Schleife durchläuft alle Produkt-Elemente -->
...
```

Schleifen

Beispiel 3.3 - Code (XSLT) (2)

```
...
        <xsl:for-each select="*">
            <!-- Diese Schleife durchläuft alle Kindelemente des Produkts -->
            <xsl:choose>
                <xsl:when test="name()='Hersteller'">
                    <p><xsl:number level="multiple" count="Produkt|Hersteller"
                        format="1.1. "/>
                    <xsl:call-template name="producer"/>
                    <b><xsl:value-of select="."/></b></p>
                </xsl:when>

                <xsl:when test="name()='Name'">
                    <i><xsl:text>Produktname: </xsl:text>
                        <xsl:value-of select="."/></i><br/>
                </xsl:when>

                <xsl:when test="name()='Farbe'">
                    <xsl:variable name="farbe"><xsl:value-of select="."/>
                    </xsl:variable>
                    <xsl:choose>
                        <xsl:when test="$farbe='blau'">
                            <i style="color: #0000FF;">
                            <xsl:value-of select="."/></i><br/>
                        </xsl:when>
...
```

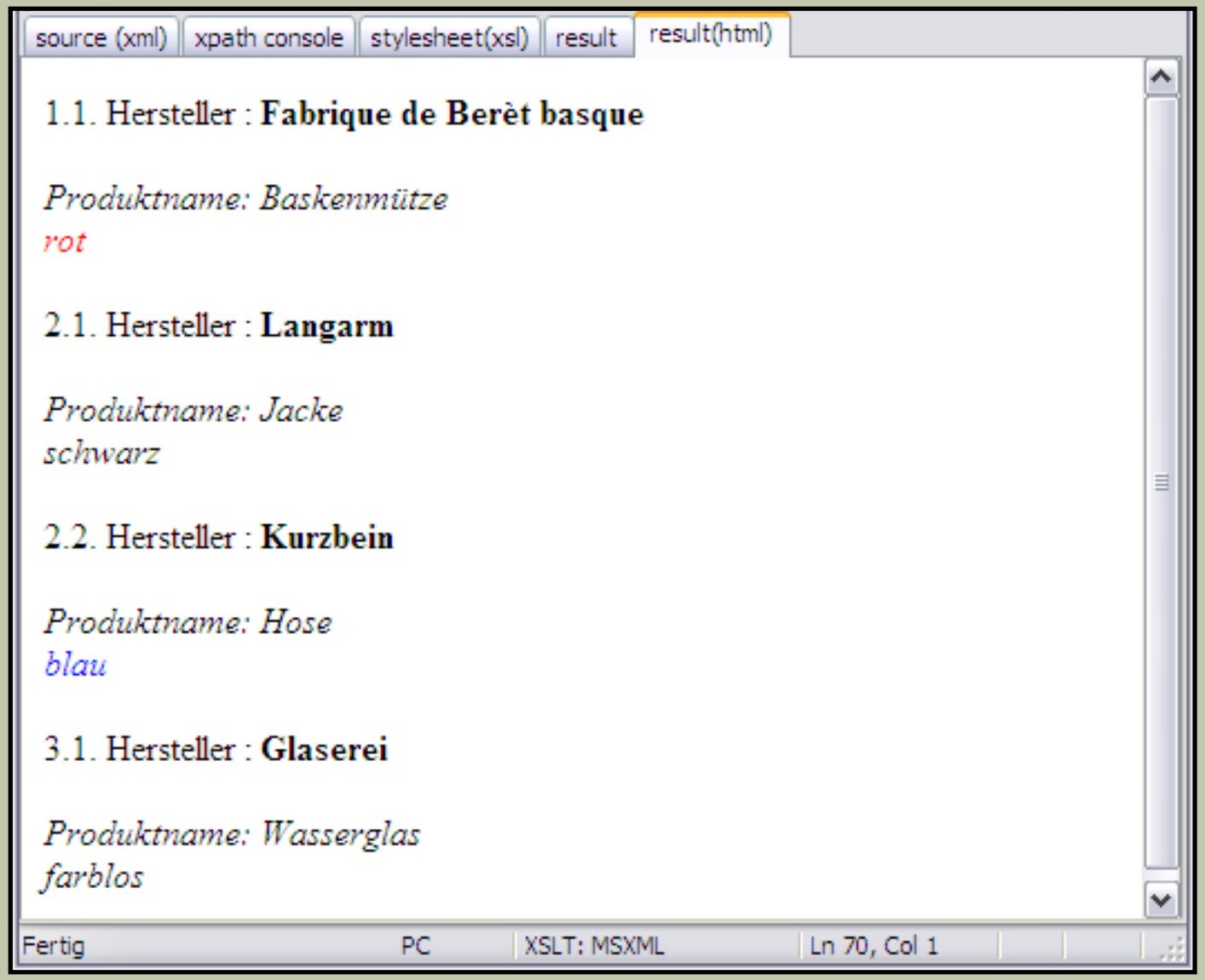
Schleifen

Beispiel 3.3 - Code (XSLT) (3)

```
...
                        <xsl:when test="$farbe='rot'">
                            <i style="color: #FF0000;">
                            <xsl:value-of select="."/></i><br/>
                        </xsl:when>
                        <xsl:otherwise>
                            <i><xsl:value-of select="."/></i><br/>
                        </xsl:otherwise>
                    </xsl:choose>
                </xsl:when>

            </xsl:choose>
        </xsl:for-each>
        <xsl:if test="not(Farbe)">
            <i>farblos</i><br/>
        </xsl:if>
    </xsl:for-each>
    </body>
    </html>
</xsl:template>

<!-- Dieses Template wird nicht für ein konkretes Element genutzt,
     sondern muss explizit aufgerufen werden. -->
...
```

Schleifen

Beispiel 3.3 - Ausgabe



Quellen

Literatur:

„XML Kompakt: Die wichtigsten Standards“

von Thilo Rottach und Sascha Groß

- Kapitel „XML-Adressierung und -Selektion“ (Seite 72/73)
- Kapitel „XML Präsentation und -Transformation“ (Seite 83-93)

„XML, XSLT, VB und ASP - Praktisches XML-Wissen für Webprojekte“

von Elmar Geese, Markus Heiliger und Matthias Lohrer

- Kapitel „Der Verwandlungskünstler XSLT“ (Seite 113-143)

WWW:

http://www.w3schools.com/Xsl/