

# Wprowadzenie do arkuszy stylistycznych XSL i transformacji XSLT

Marek Wojciechowski

[marek@cs.put.poznan.pl](mailto:marek@cs.put.poznan.pl)

<http://www.cs.put.poznan.pl/~marek/>

# Formatowanie dokumentów XML

- Język XML opisuje strukturę i semantykę, nie opisuje formatowania
  - Nie ma w XML znaczników opisujących formatowanie
  - Ze znacznikami XML nie jest związany domyślny sposób prezentacji
- Sposób prezentacji dokumentu XML (formatowanie) dodaje się poprzez dołączenie arkusza stylów
  - CSS – Cascading Style Sheets (Level 1 i Level 2)
    - Opracowany z myślą o HTML
  - XSL – Extensible Stylesheet Language
    - Opracowany dla dokumentów XML
    - Bardziej złożony, oferujący więcej możliwości niż CSS

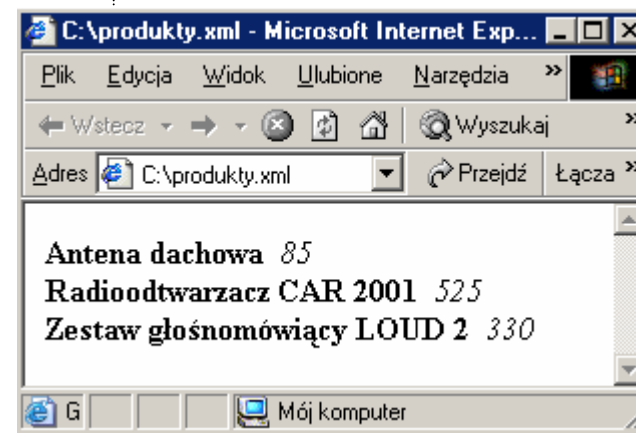
# Formatowanie XML za pomocą CSS

produkty.xml

```
<?xml version="1.0" encoding="windows-1250" ?>
<?xml-stylesheet type="text/css" href="produkty.css"?>
<cennik>
<produkt kod="67653829370">
  <nazwa>Antena dachowa</nazwa>
  <symbol>1709765</symbol>
  <cena>85</cena>
</produkt>
<produkt kod="56486294304">
  <nazwa>Radioodtworacz CAR 2001</nazwa>
  <symbol>3209765</symbol>
  <cena>525</cena>
</produkt>
<produkt kod="78488297102">
  <nazwa>Zestaw głośnomówiący LOUD 2</nazwa>
  <symbol>4409724</symbol>
  <cena>330</cena>
</produkt>
</cennik>
```

produkty.css

```
produkt {display: block}
nazwa   {display: inline;
         font-weight: bold}
symbol  {display: none}
cena    {display: inline;
         font-style: italic}
```



- CSS umożliwia formatowanie elementów, ale nie pozwala na modyfikację struktury drzewa dokumentu

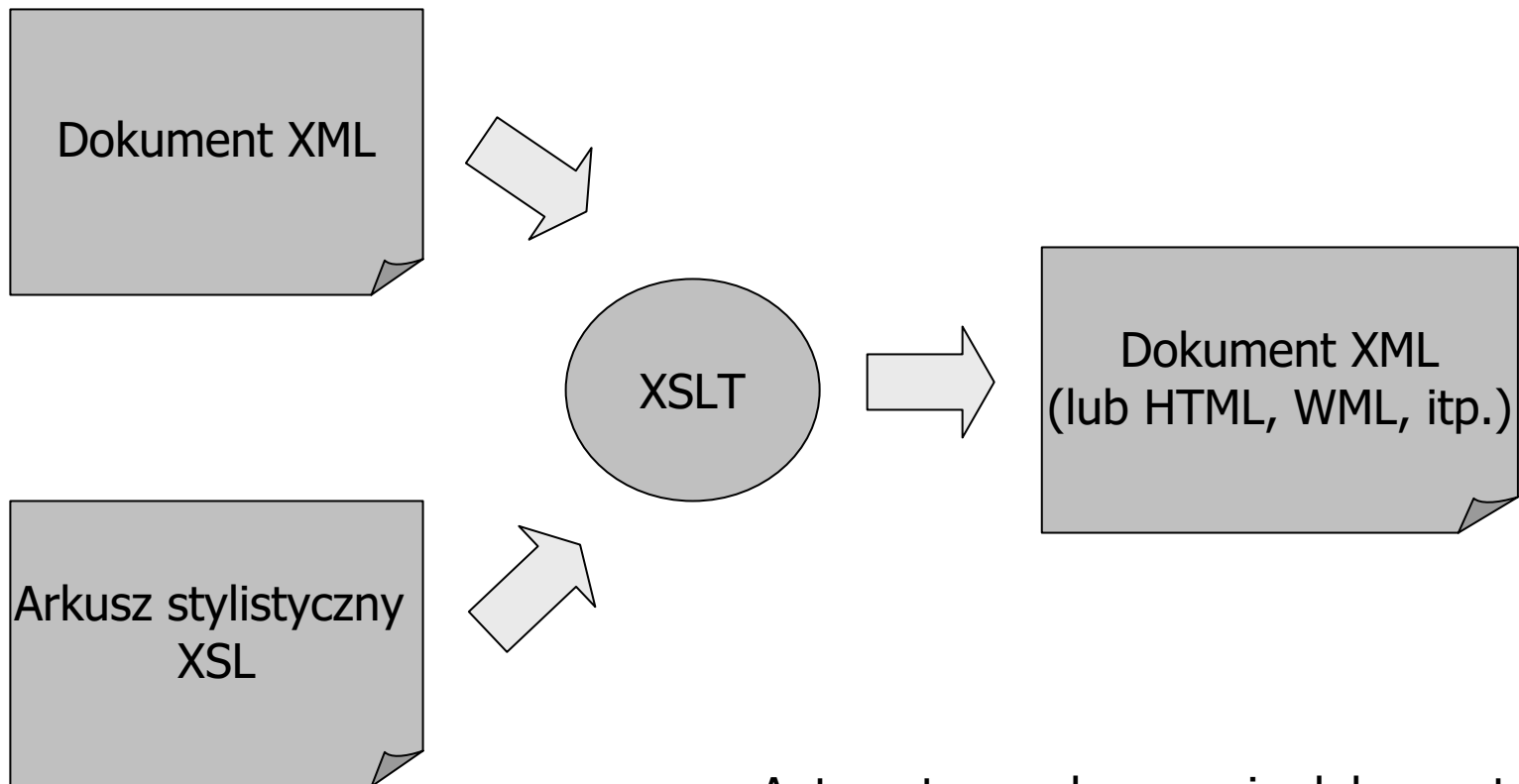
# XSL – Extensible Stylesheet Language

- Język do definiowania arkuszy stylów
- Obejmuje 2 części:
  - XSL Transformations (XSLT) – język przekształceń
    - Wykorzystuje XPath
  - XSL Formatting Objects (XSL-FO) – język opisu formatu
- W porównaniu z CSS:
  - XSL oferuje bardziej rozbudowane mechanizmy formatowania
  - Umożliwia transformację struktury drzewa dokumentu (!)
  - Arkusze XSL są dokumentami XML (!)
- XSLT może być wykorzystywany w połączeniu z XSL-FO do formatowania dokumentów XML, ale może również być wykorzystywany niezależnie jako uniwersalny język transformacji dokumentów XML

# XSLT – Język przekształceń

- Zawiera elementy służące do definiowania reguł opisujących sposób przekształcania jednego dokumentu XML na inny dokument
- Wykorzystuje język wyrażeń XPath do wyboru przetwarzanych elementów
- Przekształcony dokument może mieć znaczniki (i DTD) dokumentu oryginalnego lub używać innego zestawu znaczników
- Przykłady typów konwersji dokumentów XML:
  - Konwersja do dokumentu XML zawierającego obiekty formatujące XSL
  - Konwersja do dokumentu HTML (np. korzystającego z arkusza stylów CSS)

# Transformacja XSLT

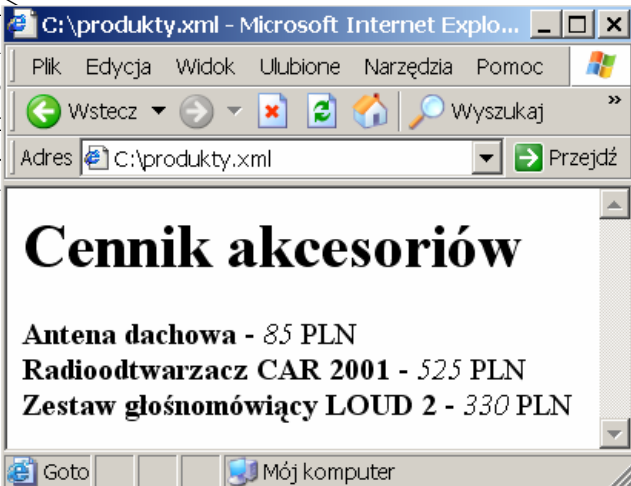


Automatyczna konwersja dokumentu XML do formatu HTML, WML, itp. lub innego dokumentu XML

# Przykład transformacji XSLT

produkty.xml

```
<?xml version="1.0"
encoding="windows-1250" ?>
<?xml-stylesheet type="text/xsl"
href="produkty.xsl"?>
<cennik>
<produkt kod="67653829370">
  <nazwa>Antena dachowa</nazwa>
  <symbol>1709765</symbol>
  <cena>85</cena>
</produkt>
<produkt kod="56486294304">
  <nazwa>Radioodtworacz CAR 2001</nazwa>
  <symbol>3209765</symbol>
  <cena>525</cena>
</produkt>
<produkt kod="78488297102">
  <nazwa>Zestaw głośnomówiący LOUD 2
</nazwa>
<symbol>3209765</symbol>
<cena>330</cena>
</produkt>
</cennik>
```



produkty.xsl

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="cennik">
    <HTML><BODY><H1>Cennik akcesoriów</H1>
    <xsl:for-each select="produkt">
      <B><xsl:value-of select="nazwa"/></B> -
      <I><xsl:value-of select="cena"/></I> PLN
    <BR/>
  </xsl:for-each>
</BODY></HTML>
</xsl:template>
</xsl:stylesheet>
```

Wyrażenia XPath wybierające przetwarzane elementy

Efekt transformacji do HTML wykonanej po stronie przeglądarki

# Metody transformacji XSLT

- Programista przygotowuje arkusz stylistyczny XSL, opisujący sposób transformacji oryginalnego dokumentu XML
- Transformacja może być opisana w sposób **rekurencyjny**, **proceduralny** lub mieszany
- Za transformację dokumentu XML odpowiada procesor XSLT
- Procesory XSLT mogą być dostępne w postaci:
  - Samodzielnych produktów
  - Modułów wchodzących w skład większych produktów
    - Przeglądarek internetowych (np. Microsoft Internet Explorer)
    - Serwerów WWW
  - Modułów bibliotecznych np. bibliotek Java

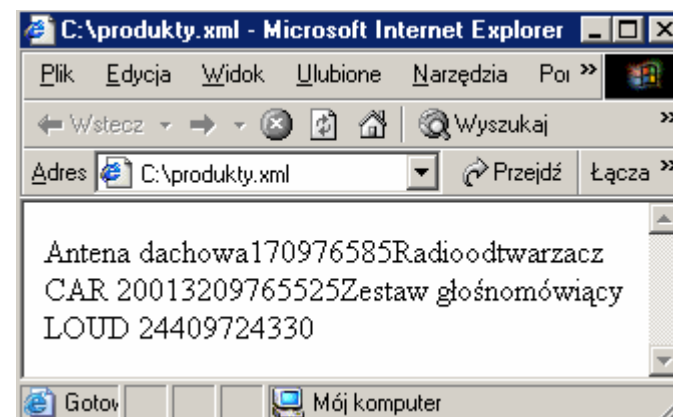
# Co widzi procesor XSLT?

- Procesor XSLT widzi drzewo dokumentu XML
- XSLT zakłada, że drzewo składa się z 7 rodzajów węzłów:
  - Korzeń
  - Elementy
  - Tekst
  - Atrybuty
  - Przestrzenie nazw
  - Instrukcje przetwarzania
  - Komentarze
- DTD i deklaracja typu dokumentu nie są włączane do drzewa (ale mogą powodować dodanie do niektórych elementów atrybutów domyślnych i stałych)

# Domyślne reguły transformacji

- XSL zawiera definicje kilku domyślnych reguł niejawnie dołączonych do wszystkich arkuszy stylów
  - Domyślna reguła elementów
    - Zapewnia, że wszystkie węzły będą przetworzone gdy nie ma jawnych reguł
  - Domyślna reguła węzłów tekstowych i atrybutów
    - Kopiuje tekst i atrybuty na wyjście
  - Domyślna reguła instrukcji przetwarzania i komentarzy
    - Pomija instrukcje przetwarzania i komentarze
- Domyślne reguły mają niższy priorytet niż wyspecyfikowane jawnie (tzn. obowiązują jeśli nie zostaną przesłonięte)
- Dzięki powyższym regułom "pusty" arkusz stylów spowoduje przekopiowanie danych tekstowych na wyjście

```
<?xml version="1.0" encoding="windows-1250"?>  
<xsl:stylesheet  
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  
  version="1.0">  
</xsl:stylesheet>
```



# Transformacja rekurencyjna (1/2)

## Przykład: Reguły transformacji

1) Każde wystąpienie znacznika `<cennik>*``</cennik>` zamień na:

```
<HTML><BODY><H1>Cennik akcesoriów</H1>*</BODY></HTML>
```

a następnie spróbuj dopasować reguły do zawartości znacznika

2) Każde wystąpienie znacznika `<produkt>*``</produkt>` zamień na:

```
*<BR/>
```

a następnie spróbuj dopasować reguły do zawartości znacznika

3) Każde wystąpienie znacznika `<nazwa>*``</nazwa>` zamień na:

```
<B>*</B> -
```

4) Każde wystąpienie znacznika `<cena>*``</cena>` zamień na:

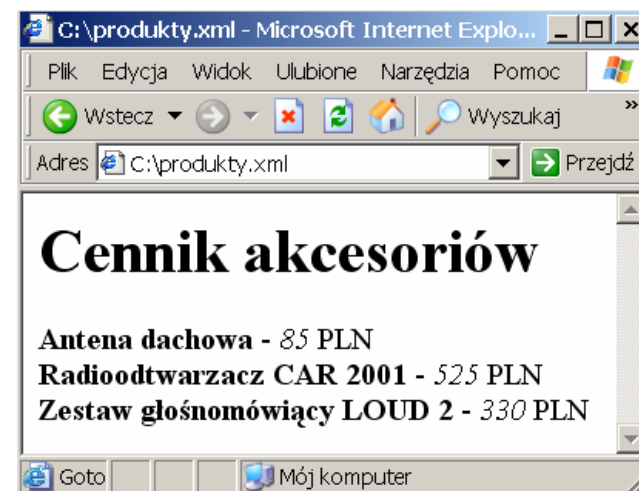
```
<I>*</I> PLN
```

5) Pomiń znacznik `<symbol>*``</symbol>`

# Transformacja rekurencyjna (2/2)

## Przykład: Arkusz stylistyczny XSL

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="cennik">
    <HTML><BODY><H1>Cennik akcesoriów</H1><xsl:apply-templates/></BODY></HTML>
  </xsl:template>
  <xsl:template match="produkt">
    <xsl:apply-templates/><BR/>
  </xsl:template>
  <xsl:template match="nazwa">
    <B><xsl:value-of select="text()" /></B> -
  </xsl:template>
  <xsl:template match="symbol">
  </xsl:template>
  <xsl:template match="cena">
    <I><xsl:value-of select="text()" /></I> PLN
  </xsl:template>
</xsl:stylesheet>
```



Wybiera węzeł tekstowy

# Transformacja proceduralna (1/2)

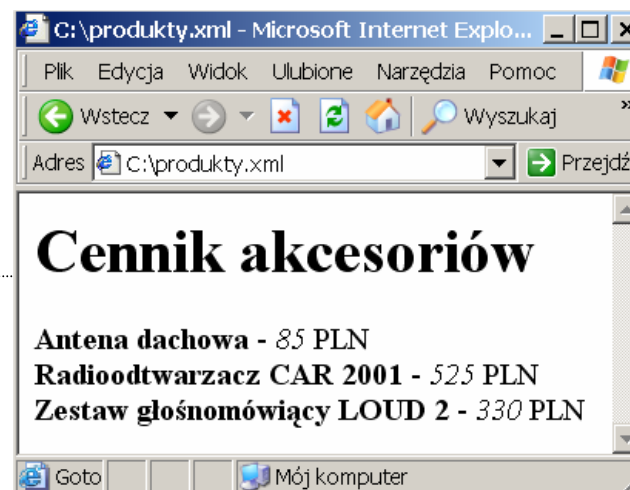
## Przykład: Pseudokod transformacji

```
wyświetl "<HTML><BODY><H1>Cennik akcesoriów</H1>";  
dla każdego znacznika <produkt> {  
    wyświetl "<B>";  
    wyświetl zawartość znacznika <nazwa>;  
    wyświetl "</B> -";  
    wyświetl "<I>";  
    wyświetl zawartość znacznika <cena>;  
    wyświetl "</I> PLN<BR/>";  
}  
wyświetl "</BODY></HTML>";
```

# Transformacja proceduralna (2/2)

## Przykład: Arkusz stylistyczny XSL

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="cennik">
    <HTML><BODY><H1>Cennik akcesoriów</H1>
    <xsl:for-each select="produkt">
      <B><xsl:value-of select="nazwa"/></B> -
      <I><xsl:value-of select="cena"/></I> PLN
    <BR/>
  </xsl:for-each>
</BODY></HTML>
</xsl:template>
</xsl:stylesheet>
```

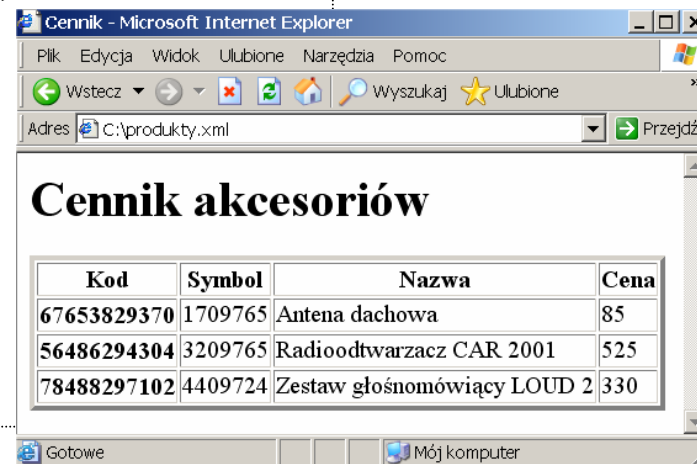


# Dostęp do atrybutów, dodawanie atrybutów

```
<?xml version="1.0" encoding="windows-1250" ?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <HTML>
      <HEAD><TITLE>Cennik</TITLE></HEAD>
      <BODY>
        <H1>Cennik akcesoriów</H1>
        <TABLE>
          <xsl:attribute name="BORDER">4</xsl:attribute>
          <TR> <TH>Kod</TH><TH>Symbol</TH>
            <TH>Nazwa</TH><TH>Cena</TH> </TR>
          <xsl:for-each select="cennik/produkt">
            <TR>
              <TH><xsl:value-of select="@kod"/></TH>
              <TD><xsl:value-of select="symbol"/></TD>
              <TD><xsl:value-of select="nazwa"/></TD>
              <TD><xsl:value-of select="cena"/></TD>
            </TR>
          </xsl:for-each>
        </TABLE>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

Dodanie atrybutu *BORDER* z wartością 4 do elementu *<TABLE>* (*<TABLE BORDER="4">* też OK.)

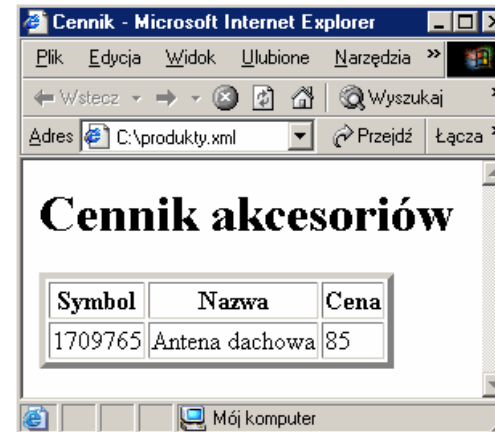
Dostęp do atrybutu *KOD* elementu *cennik/produkt*



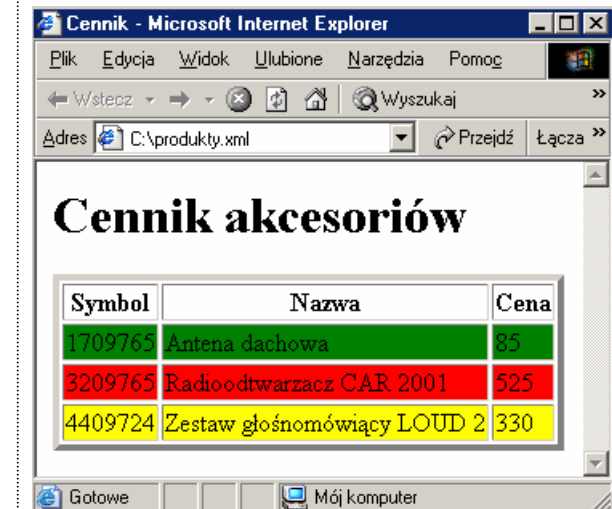
Kod	Symbol	Nazwa	Cena
67653829370	1709765	Antena dachowa	85
56486294304	3209765	Radiodtwarzacz CAR 2001	525
78488297102	4409724	Zestaw głośnomówiący LOUD 2	330

# Transformacja warunkowa w XSLT

```
<xsl:for-each select="cennik/produkt">
  <xsl:if test="symbol='1709765'">
    <TR>
      <TD><xsl:value-of select="symbol"/></TD>
      <TD><xsl:value-of select="nazwa"/></TD>
      <TD><xsl:value-of select="cena"/></TD>
    </TR>
  </xsl:if>
</xsl:for-each>
```

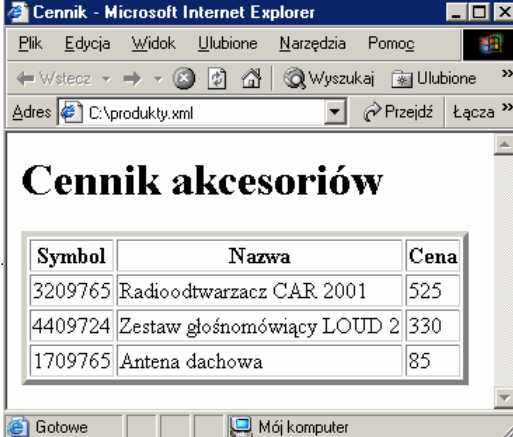


```
<xsl:for-each select="cennik/produkt">
  <TR>
    <xsl:attribute name="BGCOLOR">
      <xsl:choose>
        <xsl:when test="symbol='1709765'">green</xsl:when>
        <xsl:when test="symbol='4409724'">yellow</xsl:when>
        <xsl:otherwise>red</xsl:otherwise>
      </xsl:choose>
    </xsl:attribute>
    <TD><xsl:value-of select="symbol"/></TD>
    <TD><xsl:value-of select="nazwa"/></TD>
    <TD><xsl:value-of select="cena"/></TD>
  </TR>
</xsl:for-each>
```



# Sortowanie w XSLT

```
<xsl:for-each select="cennik/produkt">  
  <xsl:sort select="cena" data-type="number" order="descending"/>  
  <TR>  
    <TD><xsl:value-of select="symbol"/></TD>  
    <TD><xsl:value-of select="nazwa"/></TD>  
    <TD><xsl:value-of select="cena"/></TD>  
  </TR>  
</xsl:for-each>
```



Cennik - Microsoft Internet Explorer

Plik Edycja Widok Ulubione Narzędzia Pomoc

Wstecz Wyszukaj Ulubione

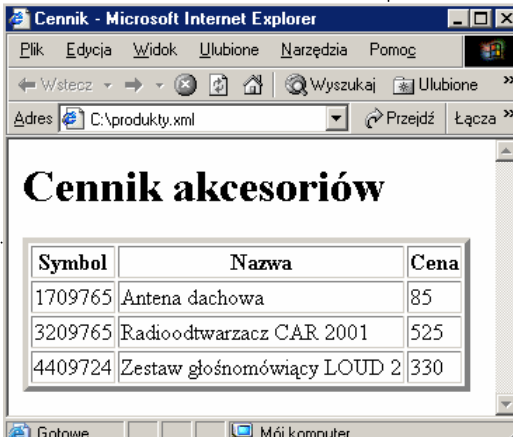
Adres C:\produkty.xml Przejdź Łącz

### Cennik akcesoriów

Symbol	Nazwa	Cena
3209765	Radioodtwarzacz CAR 2001	525
4409724	Zestaw głośnomówiący LOUD 2	330
1709765	Antena dachowa	85

Gotowe Mój komputer

```
<xsl:for-each select="cennik/produkt">  
  <xsl:sort select="nazwa" lang="pl" data-type="text" order="ascending"/>  
  <xsl:sort select="symbol" data-type="number" order="ascending"/>  
  <TR>  
    <TD><xsl:value-of select="symbol"/></TD>  
    <TD><xsl:value-of select="nazwa"/></TD>  
    <TD><xsl:value-of select="cena"/></TD>  
  </TR>  
</xsl:for-each>
```



Cennik - Microsoft Internet Explorer

Plik Edycja Widok Ulubione Narzędzia Pomoc

Wstecz Wyszukaj Ulubione

Adres C:\produkty.xml Przejdź Łącz

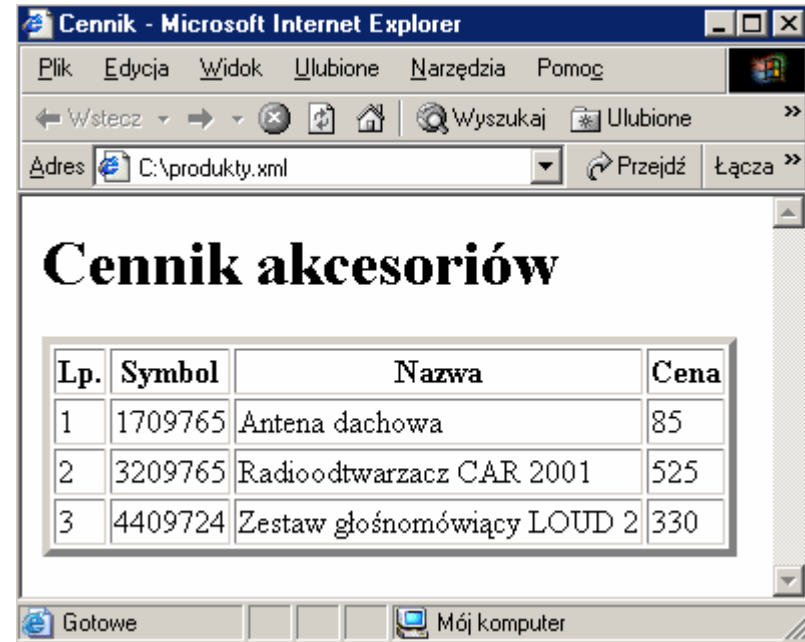
### Cennik akcesoriów

Symbol	Nazwa	Cena
1709765	Antena dachowa	85
3209765	Radioodtwarzacz CAR 2001	525
4409724	Zestaw głośnomówiący LOUD 2	330

Gotowe Mój komputer

# Automatyczne numerowanie w XSLT

```
<xsl:for-each select="cennik/produkt">
  <TR>
    <TD><xsl:number/></TD>
    <TD><xsl:value-of select="symbol"/></TD>
    <TD><xsl:value-of select="nazwa"/></TD>
    <TD><xsl:value-of select="cena"/></TD>
  </TR>
</xsl:for-each>
```



Cennik - Microsoft Internet Explorer

Pluk Edycja Widok Ulubione Narzędzia Pomoc

Wstecz Wyszukaj Ulubione

Adres C:\produkty.xml

## Cennik akcesoriów

Lp.	Symbol	Nazwa	Cena
1	1709765	Antena dachowa	85
2	3209765	Radioodtworacz CAR 2001	525
3	4409724	Zestaw głośnomówiący LOUD 2	330

Gotowe Mój komputer

- Domyślnie **xsl:number** zlicza sąsiadujące węzły węzła źródłowego
- Możliwości zmiany domyślnego zachowania:
  - `<xsl:number value="4"/>`: wartość wyrażenia
  - `<xsl:number level="any"/>`: numeracja elementów danego typu w dokumencie
  - `<xsl:number ... format="I"/>`: I-rzymskie, i-rzymskie małe, a/A - litery
  - ...

# Metody wyprowadzania wyniku

- Dostępne metody: **xml**, **html**, **text**
- Metodę określa element **xsl:output** np. `<xsl:output method="xml">`
- Domyślnie **xml**, chyba że element główny dokumentu wynikowego to `<HTML></HTML>`
- Przy generacji HTML metodą **xml** dokument musi być well-formed:
  - Znaczniki muszą być zamykane; np. `<P>...</P>`; znaczniki puste w notacji XML; np. `<BR/>`
  - Znaczniki nie mogą się nakładać; np. `<B> raz <I> dwa </B> trzy </I>` jest błędem
  - Wielkość liter istotna; np. `<B><i> raz </I></B>` jest błędem
  - Wartości atrybutów w cudzysłowach
  - Cały dokument zawarty w `<HTML>...</HTML>`

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="html" encoding="windows-1250"/>
  <xsl:template match="cennik">
    <HTML> ... </HTML>
  </xsl:template>
</xsl:stylesheet>
```

# Łączenie kilku arkuszy stylów

- Importowanie – **xml:import**
  - Może wystąpić tylko w elemencie głównym – **xsl:stylesheet** – przed innymi elementami
  - Reguły z importowanego arkusza mają niższy priorytet niż reguły zdefiniowane lokalnie
- Włączanie – **xml:include**
  - Może wystąpić gdziekolwiek w elemencie głównym po ostatnim elemencie **xsl:import**
  - Reguły z włączanego arkusza mają taki sam priorytet jak reguły zdefiniowane lokalnie – są traktowane tak jak gdyby były zdefiniowane w miejscu włączenia

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:import href="importowany1.xsl"/>
<xsl:import href="importowany2.xsl"/>
...
<xsl:include href="wlacznany.xsl"/>
...
</xsl:stylesheet>
```