Programowanie komunikacji człowiek-komputer

dr inż. Joanna Ochelska-Mierzejewska

2

XSL
Extensible Stylesheet Language

pol. rozszerzalny język arkuszy stylów

3

Extensible Stylesheet Language (XSL)

- Zdefiniowany w rekomendacjach (wersje 1.0 w 1999 i 2001)
  - **XSL** (ogólne ramy, język XSL Formatting Objects)
  - **XSLT** (arkusze – przekształcenia dokumentów XML)
  - **XPath** (język wyrażeń zawierający ścieżki do adresowania fragmentów dokumentu)
- Arkusz stylu mówi, jak przekształcać dany typ dokumentu do dokumentu XSL-FO
- W praktyce przekształcenia także do innych formatów, często (X)HTML
- Zazwyczaj dany arkusz jest odpowiedni dla dokumentów XML określonego rodzaju (konkretnego zastosowania XML)

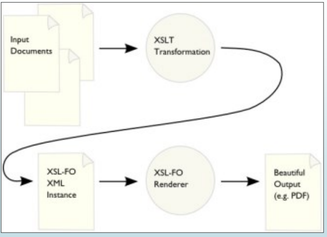
4

Rola XPath w XSLT

- Dostęp do dokumentu źródłowego możliwy dzięki **ścieżkom XPath**
- Możliwe
  - umieszczenie fragmentu dokumentu lub wartości odczytanej z dokumentu
  - umieszczenie wartości „obliczonej" wyrażeniem XPath
  - sprawdzenie warunku logicznego
- XSLT 1.0 używa XPath w wersji 1.0
- XSLT 2.0 używa XPath w wersji 2.0

5

Idea XSL

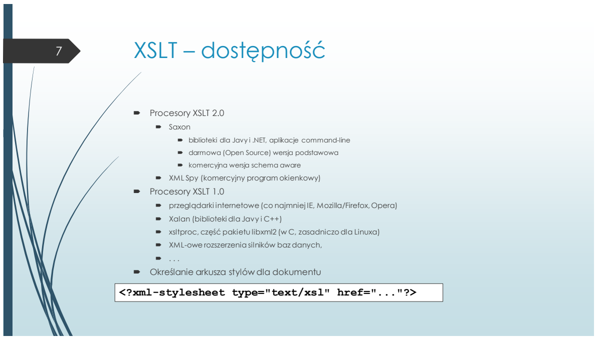


Źródło: W3C, Rekomendacja XSL

6

XSLT – status

- Powstał w ramach standardu XSL
- Zastosowania wykraczają poza wizualizację XML
- Wersja 1.0
  - listopad 1999, powiązane z XPath 1.0
  - szerokie wsparcie w oprogramowaniu
- Wersja 2.0
  - styczeń 2007, powiązane z XPath 2.0 i XQuery 1.0
  - głębsze podstawy teoretyczne, większe możliwości
  - mniejsze (ale istniejące) wsparcie

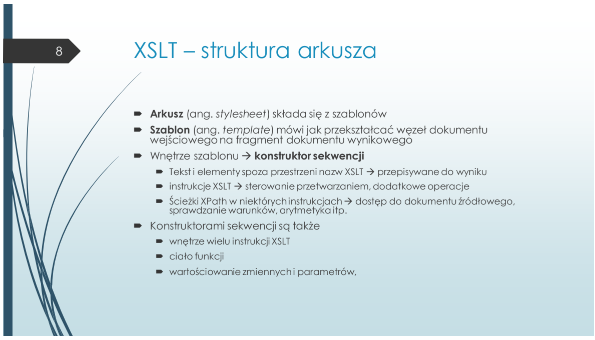


7

XSLT – dostępność

- Procesory XSLT 2.0
  - Saxon
    - biblioteki dla Javy i .NET, aplikacje command-line
    - darmowa (Open Source) wersja podstawowa
    - komercyjna wersja schema aware
  - XML Spy (komercyjny program okienkowy)
- Procesory XSLT 1.0
  - przeglądarki internetowe (co najmniej IE, Mozilla/Firefox, Opera)
  - Xalan (biblioteki dla Javy i C++)
  - xsltproc, część pakietu libxml2 (w C, zasadniczo dla Linuxa)
  - XML-owe rozszerzenia silników baz danych,
  - . . .
- Określanie arkusza stylów dla dokumentu

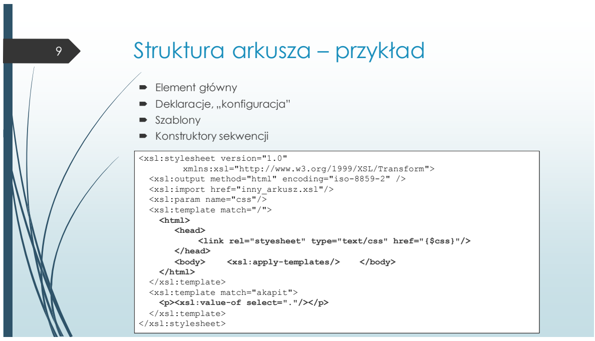
```
<?xml-stylesheet type="text/xsl" href="..."?>
```



8

XSLT – struktura arkusza

- **Arkusz** (ang. *stylesheet*) składa się z szablonów
- **Szablon** (ang. *template*) mówi jak przekształcać węzeł dokumentu wejściowego na fragment dokumentu wynikowego
- Wnętrze szablonu → **konstruktor sekwencji**
  - Tekst i elementy spoza przestrzeni nazw XSLT → przepisywane do wyniku
  - instrukcje XSLT → sterowanie przetwarzaniem, dodatkowe operacje
  - Ścieżki XPath w niektórych instrukcjach → dostęp do dokumentu źródłowego, sprawdzanie warunków, arytmetyka itp.
- Konstruktorami sekwencji są także
  - wnętrze wielu instrukcji XSLT
  - ciało funkcji
  - wartościowanie zmiennych i parametrów,

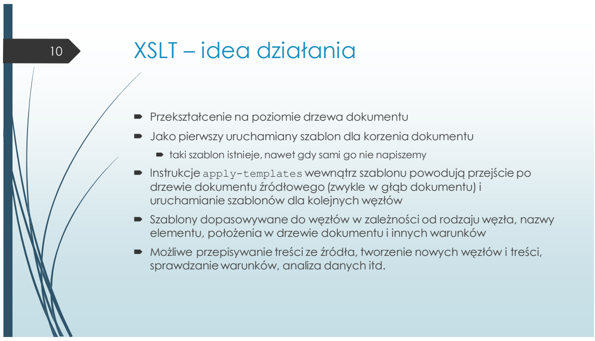


9

Struktura arkusza – przykład

- Element główny
- Deklaracje, „konfiguracja"
- Szablony
- Konstruktory sekwencji

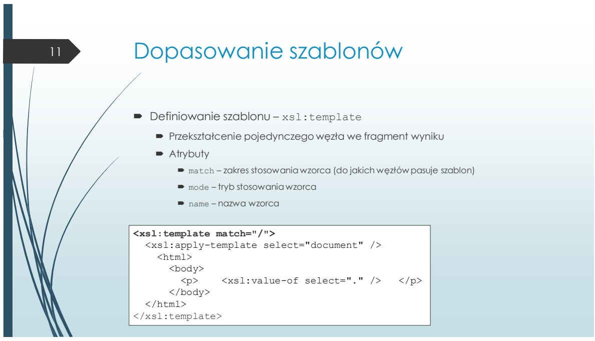
```
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" encoding="iso-8859-2" />
  <xsl:import href="inny_arkusz.xsl"/>
  <xsl:param name="css"/>
  <xsl:template match="/">
    <html>
       <head>
            <link rel="styesheet" type="text/css" href="{$css}"/>
       </head>
       <body>    <xsl:apply-templates/>    </body>
    </html>
  </xsl:template>
  <xsl:template match="akapit">
    <p><xsl:value-of select="."/></p>
  </xsl:template>
</xsl:stylesheet>
```



10

XSLT – idea działania

- Przekształcenie na poziomie drzewa dokumentu
- Jako pierwszy uruchamiany szablon dla korzenia dokumentu
  - taki szablon istnieje, nawet gdy sami go nie napiszemy
- Instrukcje `apply-templates` wewnątrz szablonu powodują przejście po drzewie dokumentu źródłowego (zwykle w głąb dokumentu) i uruchamianie szablonów dla kolejnych węzłów
- Szablony dopasowywane do węzłów w zależności od rodzaju węzła, nazwy elementu, położenia w drzewie dokumentu i innych warunków
- Możliwe przepisywanie treści ze źródła, tworzenie nowych węzłów i treści, sprawdzanie warunków, analiza danych itd.

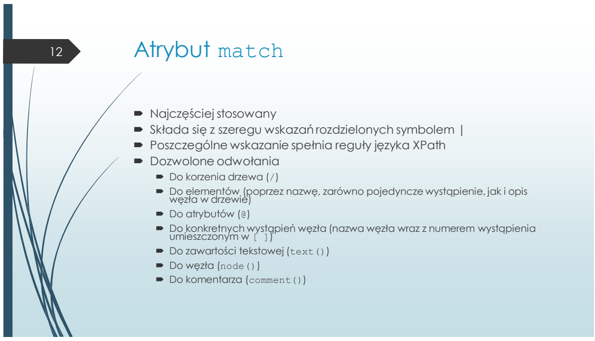


11

Dopasowanie szablonów

- Definiowanie szablonu – `xsl:template`
  - Przekształcenie pojedynczego węzła we fragment wyniku
  - Atrybuty
    - `match` – zakres stosowania wzorca (do jakich węzłów pasuje szablon)
    - `mode` – tryb stosowania wzorca
    - `name` – nazwa wzorca

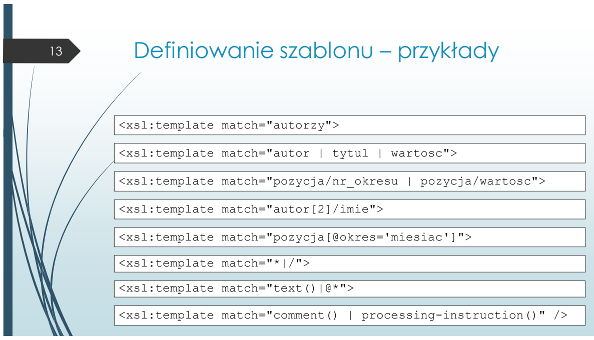
```
<xsl:template match="/">
  <xsl:apply-template select="document" />
    <html>
      <body>
        <p>    <xsl:value-of select="." />    </p>
      </body>
  </html>
</xsl:template>
```



12

Atrybut `match`

- Najczęściej stosowany
- Składa się z szeregu wskazań rozdzielonych symbolem |
- Poszczególne wskazanie spełnia reguły języka XPath
- Dozwolone odwołania
  - Do korzenia drzewa (`/`)
  - Do elementów (poprzez nazwę, zarówno pojedyncze wystąpienie, jak i opis węzła w drzewie)
  - Do atrybutów (`@`)
  - Do konkretnych wystąpień węzła (nazwa węzła wraz z numerem wystąpienia umieszczonym w `[ ]`)
  - Do zawartości tekstowej (`text()`)
  - Do węzła (`node()`)
  - Do komentarza (`comment()`)



13

Definiowanie szablonu – przykłady

```
<xsl:template match="autorzy">
```

```
<xsl:template match="autor | tytul | wartosc">
```

```
<xsl:template match="pozycja/nr_okresu | pozycja/wartosc">
```

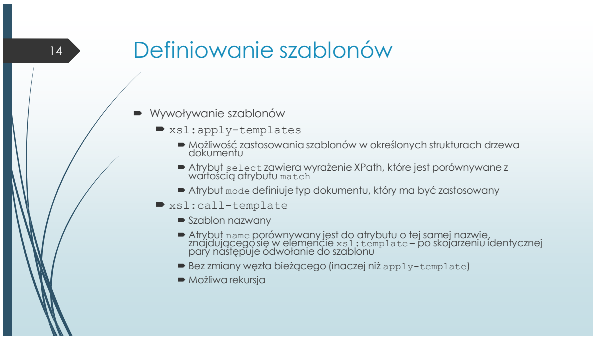
```
<xsl:template match="autor[2]/imie">
```

```
<xsl:template match="pozycja[@okres='miesiac']">
```

```
<xsl:template match="*|/">
```

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

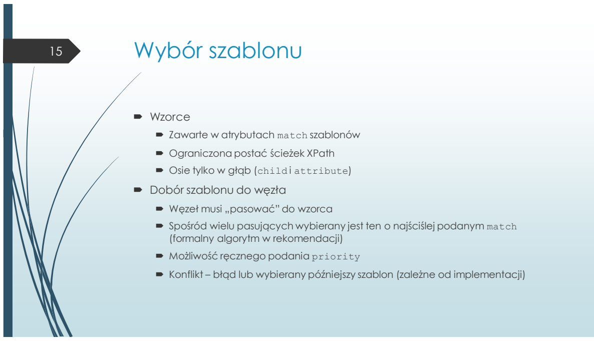
```
<xsl:template match="comment() | processing-instruction()" />
```



14

Definiowanie szablonów

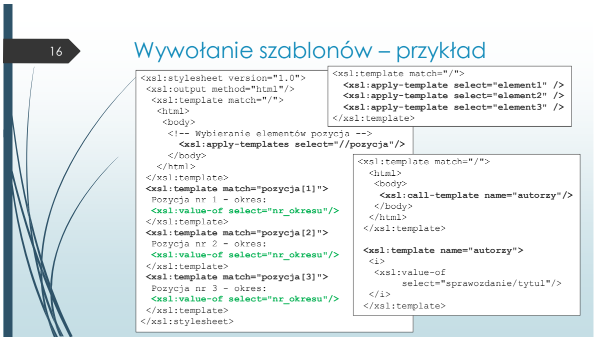
- Wywoływanie szablonów
  - `xsl:apply-templates`
    - Możliwość zastosowania szablonów w określonych strukturach drzewa dokumentu
    - Atrybut `select` zawiera wyrażenie XPath, które jest porównywane z wartością atrybutu `match`
    - Atrybut `mode` definiuje typ dokumentu, który ma być zastosowany
  - `xsl:call-template`
    - Szablon nazwany
    - Atrybut `name` porównywany jest do atrybutu o tej samej nazwie, znajdującego się w elemencie `xsl:template` – po skojarzeniu identycznej pary następuje odwołanie do szablonu
    - Bez zmiany węzła bieżącego (inaczej niż `apply-template`)
    - Możliwa rekursja



15

Wybór szablonu

- Wzorce
  - Zawarte w atrybutach `match` szablonów
  - Ograniczona postać ścieżek XPath
  - Osie tylko w głąb (`child` i `attribute`)
- Dobór szablonu do węzła
  - Węzeł musi „pasować" do wzorca
  - Spośród wielu pasujących wybierany jest ten o najściślej podanym `match` (formalny algorytm w rekomendacji)
  - Możliwość ręcznego podania `priority`
  - Konflikt – błąd lub wybierany późniejszy szablon (zależne od implementacji)



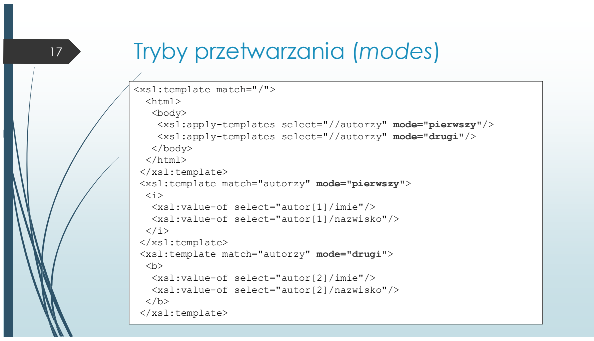
16

Wywołanie szablonów – przykład

```
<xsl:stylesheet version="1.0">
 <xsl:output method="html"/>
  <xsl:template match="/">
   <html>
    <body>
     <!-- Wybieranie elementów pozycja -->
       <xsl:apply-templates select="//pozycja"/>
     </body>
   </html>
 </xsl:template>
 <xsl:template match="pozycja[1]">
  Pozycja nr 1 - okres:
  <xsl:value-of select="nr_okresu"/>
 </xsl:template>
 <xsl:template match="pozycja[2]">
  Pozycja nr 2 - okres:
  <xsl:value-of select="nr_okresu"/>
 </xsl:template>
 <xsl:template match="pozycja[3]">
  Pozycja nr 3 - okres:
  <xsl:value-of select="nr_okresu"/>
 </xsl:template>
</xsl:stylesheet>
```

```
<xsl:template match="/">
  <xsl:apply-template select="element1" />
  <xsl:apply-template select="element2" />
  <xsl:apply-template select="element3" />
</xsl:template>
```

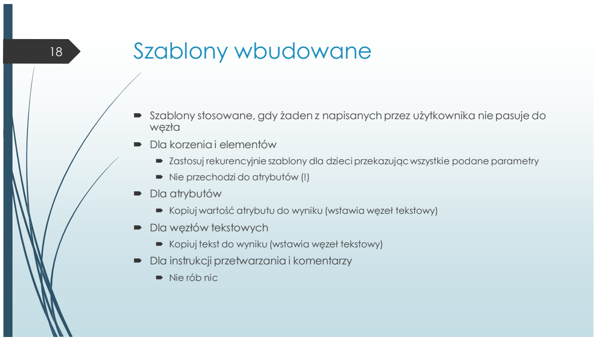
```
<xsl:template match="/">
  <html>
   <body>
    <xsl:call-template name="autorzy"/>
   </body>
  </html>
 </xsl:template>

 <xsl:template name="autorzy">
  <i>
   <xsl:value-of
        select="sprawozdanie/tytul"/>
  </i>
 </xsl:template>
```



17

Tryby przetwarzania (*modes*)

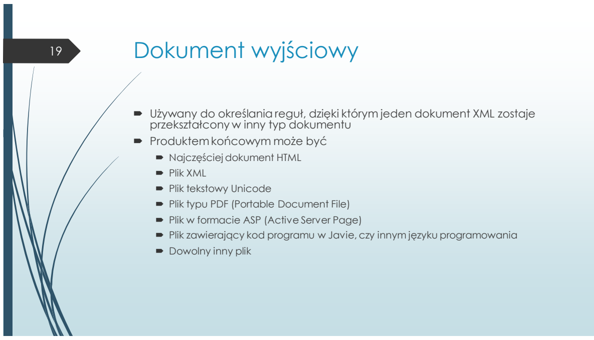
```
<xsl:template match="/">
  <html>
   <body>
    <xsl:apply-templates select="//autorzy" mode="pierwszy"/>
    <xsl:apply-templates select="//autorzy" mode="drugi"/>
   </body>
  </html>
 </xsl:template>
 <xsl:template match="autorzy" mode="pierwszy">
  <i>
   <xsl:value-of select="autor[1]/imie"/>
   <xsl:value-of select="autor[1]/nazwisko"/>
  </i>
 </xsl:template>
 <xsl:template match="autorzy" mode="drugi">
  <b>
   <xsl:value-of select="autor[2]/imie"/>
   <xsl:value-of select="autor[2]/nazwisko"/>
  </b>
 </xsl:template>
```



18

Szablony wbudowane

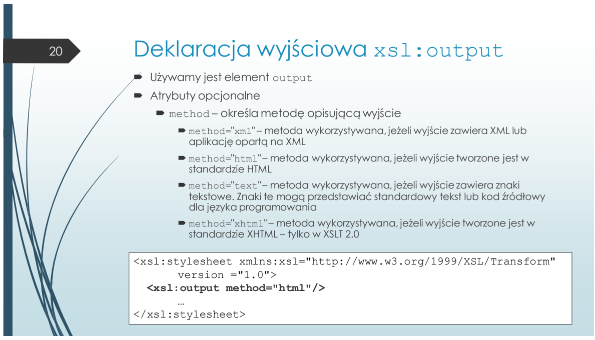
- Szablony stosowane, gdy żaden z napisanych przez użytkownika nie pasuje do węzła
- Dla korzenia i elementów
  - Zastosuj rekurencyjnie szablony dla dzieci przekazując wszystkie podane parametry
  - Nie przechodzi do atrybutów (!)
- Dla atrybutów
  - Kopiuj wartość atrybutu do wyniku (wstawia węzeł tekstowy)
- Dla węzłów tekstowych
  - Kopiuj tekst do wyniku (wstawia węzeł tekstowy)
- Dla instrukcji przetwarzania i komentarzy
  - Nie rób nic



19

Dokument wyjściowy

- Używany do określania reguł, dzięki którym jeden dokument XML zostaje przekształcony w inny typ dokumentu
- Produktem końcowym może być
  - Najczęściej dokument HTML
  - Plik XML
  - Plik tekstowy Unicode
  - Plik typu PDF (Portable Document File)
  - Plik w formacie ASP (Active Server Page)
  - Plik zawierający kod programu w Javie, czy innym języku programowania
  - Dowolny inny plik

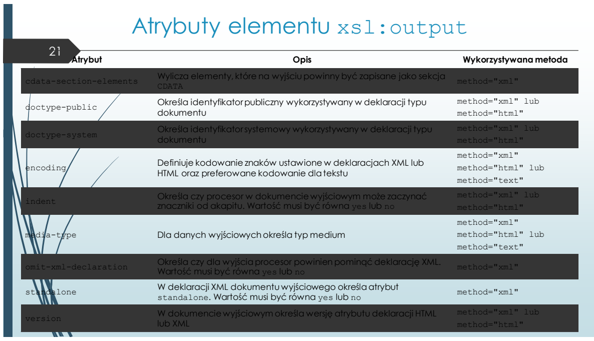


20

Deklaracja wyjściowa `xsl:output`

- Używamy jest element `output`
- Atrybuty opcjonalne
  - `method` – określa metodę opisującą wyjście
    - `method="xml"` – metoda wykorzystywana, jeżeli wyjście zawiera XML lub aplikację opartą na XML
    - `method="html"` – metoda wykorzystywana, jeżeli wyjście tworzone jest w standardzie HTML
    - `method="text"` – metoda wykorzystywana, jeżeli wyjście zawiera znaki tekstowe. Znaki te mogą przedstawiać standardowy tekst lub kod źródłowy dla języka programowania
    - `method="xhtml"` – metoda wykorzystywana, jeżeli wyjście tworzone jest w standardzie XHTML – tylko w XSLT 2.0

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       version ="1.0">
  <xsl:output method="html"/>
       …
</xsl:stylesheet>
```



Atrybuty elementu `xsl:output`

21

Atrybut	Opis	Wykorzystywana metoda
`cdata-section-elements`	Wylicza elementy, które na wyjściu powinny być zapisane jako sekcja `CDATA`	`method="xml"`

`doctype-public`	Określa identyfikator publiczny wykorzystywany w deklaracji typu dokumentu	`method="xml" lub`
		`method="html"`

`doctype-system`	Określa identyfikator systemowy wykorzystywany w deklaracji typu dokumentu	`method="xml" lub`
		`method="html"`

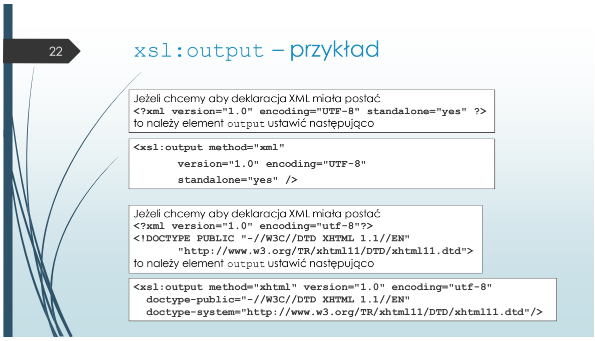
`encoding`	Definiuje kodowanie znaków ustawione w deklaracjach XML lub HTML oraz preferowane kodowanie dla tekstu	`method="xml"`
		`method="html" lub`
		`method="text"`

`indent`	Określa czy procesor w dokumencie wyjściowym może zaczynać znaczniki od akapitu. Wartość musi być równa `yes` lub `no`	`method="xml" lub`
		`method="html"`

`media-type`	Dla danych wyjściowych określa typ medium	`method="xml"`
		`method="html" lub`
		`method="text"`

`omit-xml-declaration`	Określa czy dla wyjścia procesor powinien pominąć deklarację XML. Wartość musi być równa `yes` lub `no`	`method="xml"`
`standalone`	W deklaracji XML dokumentu wyjściowego określa atrybut `standalone`. Wartość musi być równa `yes` lub `no`	`method="xml"`

`version`	W dokumencie wyjściowym określa wersję atrybutu deklaracji HTML lub XML	`method="xml" lub`
		`method="html"`



22

`xsl:output` – przykład

Jeżeli chcemy aby deklaracja XML miała postać
`<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>`
to należy element `output` ustawić następująco

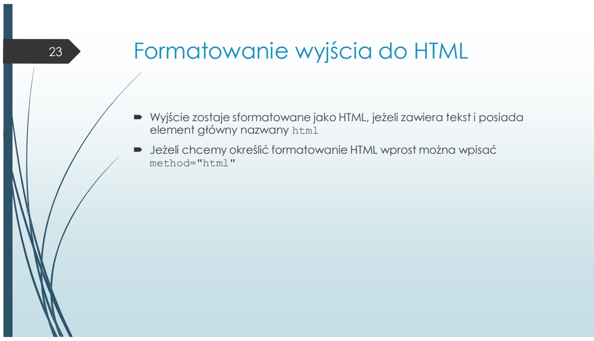
```
<xsl:output method="xml"
       version="1.0" encoding="UTF-8"
       standalone="yes" />
```

Jeżeli chcemy aby deklaracja XML miała postać

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE PUBLIC "-//W3C//DTD XHTML 1.1//EN"
       "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

to należy element `output` ustawić następująco

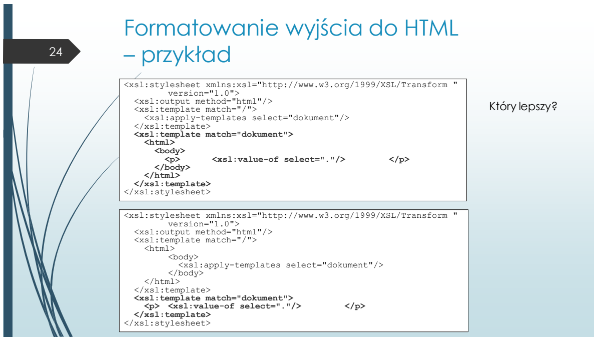
```
<xsl:output method="xhtml" version="1.0" encoding="utf-8"
  doctype-public="-//W3C//DTD XHTML 1.1//EN"
  doctype-system="http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"/>
```



23

Formatowanie wyjścia do HTML

- Wyjście zostaje sformatowane jako HTML, jeżeli zawiera tekst i posiada element główny nazwany `html`
- Jeżeli chcemy określić formatowanie HTML wprost można wpisać `method="html"`



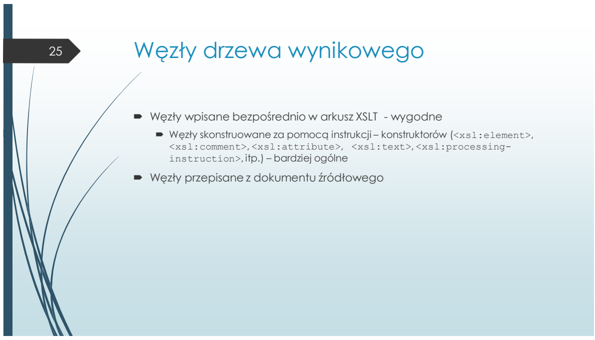
24

Formatowanie wyjścia do HTML – przykład

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform "
        version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <xsl:apply-templates select="dokument"/>
  </xsl:template>
  <xsl:template match="dokument">
    <html>
      <body>
        <p>     <xsl:value-of select="."/>        </p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform "
        version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
        <body>
          <xsl:apply-templates select="dokument"/>
        </body>
    </html>
  </xsl:template>
  <xsl:template match="dokument">
    <p>  <xsl:value-of select="."/>        </p>
  </xsl:template>
</xsl:stylesheet>
```

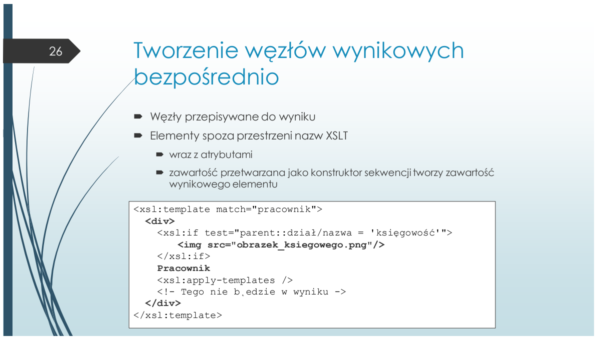
Który lepszy?



25

Węzły drzewa wynikowego

- Węzły wpisane bezpośrednio w arkusz XSLT - wygodne
  - Węzły skonstruowane za pomocą instrukcji – konstruktorów (`<xsl:element>`, `<xsl:comment>`, `<xsl:attribute>`, `<xsl:text>`, `<xsl:processing-instruction>`, itp.) – bardziej ogólne
- Węzły przepisane z dokumentu źródłowego

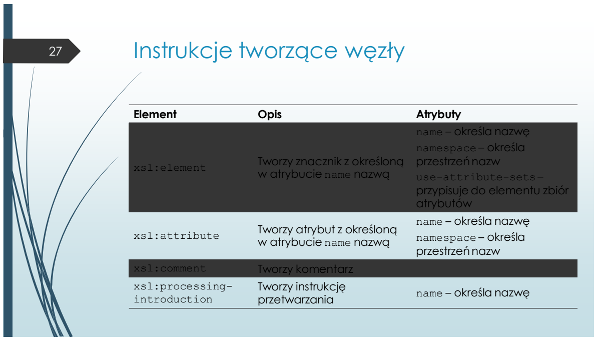


26

Tworzenie węzłów wynikowych bezpośrednio

- Węzły przepisywane do wyniku
- Elementy spoza przestrzeni nazw XSLT
  - wraz z atrybutami
  - zawartość przetwarzana jako konstruktor sekwencji tworzy zawartość wynikowego elementu

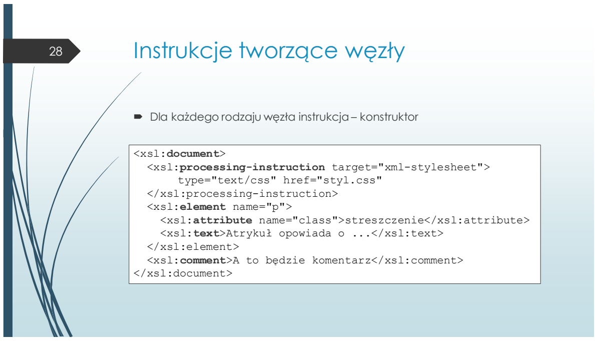
```
<xsl:template match="pracownik">
  <div>
    <xsl:if test="parent::dział/nazwa = 'księgowość'">
       <img src="obrazek_ksiegowego.png"/>
    </xsl:if>
    Pracownik
    <xsl:apply-templates />
    <!- Tego nie b˛edzie w wyniku ->
  </div>
</xsl:template>
```



27

Instrukcje tworzące węzły

Element	Opis	Atrybuty
`xsl:element`	Tworzy znacznik z określoną w atrybucie `name` nazwą	`name` – określa nazwę `namespace` – określa przestrzeń nazw `use-attribute-sets` – przypisuje do elementu zbiór atrybutów
`xsl:attribute`	Tworzy atrybut z określoną w atrybucie `name` nazwą	`name` – określa nazwę `namespace` – określa przestrzeń nazw
`xsl:comment`	Tworzy komentarz	
`xsl:processing-introduction`	Tworzy instrukcję przetwarzania	`name` – określa nazwę

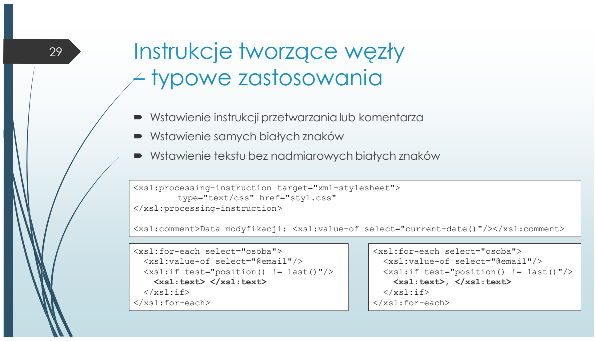


28

Instrukcje tworzące węzły

- Dla każdego rodzaju węzła instrukcja – konstruktor

```
<xsl:document>
  <xsl:processing-instruction target="xml-stylesheet">
       type="text/css" href="styl.css"
  </xsl:processing-instruction>
  <xsl:element name="p">
    <xsl:attribute name="class">streszczenie</xsl:attribute>
    <xsl:text>Atrykuł opowiada o ...</xsl:text>
  </xsl:element>
  <xsl:comment>A to będzie komentarz</xsl:comment>
</xsl:document>
```



29

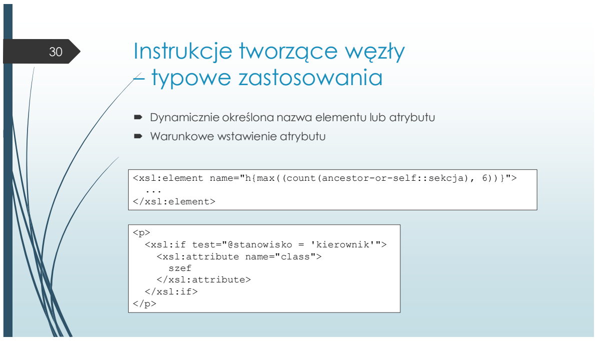
Instrukcje tworzące węzły – typowe zastosowania

- Wstawienie instrukcji przetwarzania lub komentarza
- Wstawienie samych białych znaków
- Wstawienie tekstu bez nadmiarowych białych znaków

```
<xsl:processing-instruction target="xml-stylesheet">
        type="text/css" href="styl.css"
</xsl:processing-instruction>

<xsl:comment>Data modyfikacji: <xsl:value-of select="current-date()"/></xsl:comment>
```

```
<xsl:for-each select="osoba">
  <xsl:value-of select="@email"/>
  <xsl:if test="position() != last()"/>
    <xsl:text> </xsl:text>
  </xsl:if>
</xsl:for-each>
```

```
<xsl:for-each select="osoba">
  <xsl:value-of select="@email"/>
  <xsl:if test="position() != last()"/>
    <xsl:text>, </xsl:text>
  </xsl:if>
</xsl:for-each>
```



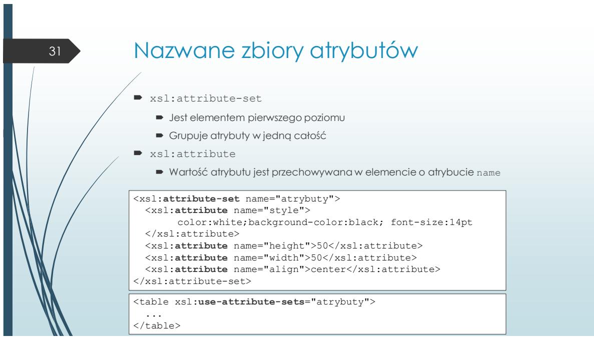
30

Instrukcje tworzące węzły – typowe zastosowania

- Dynamicznie określona nazwa elementu lub atrybutu
- Warunkowe wstawienie atrybutu

```
<xsl:element name="h{max((count(ancestor-or-self::sekcja), 6))}">
  ...
</xsl:element>
```

```
<p>
  <xsl:if test="@stanowisko = 'kierownik'">
    <xsl:attribute name="class">
      szef
    </xsl:attribute>
  </xsl:if>
</p>
```



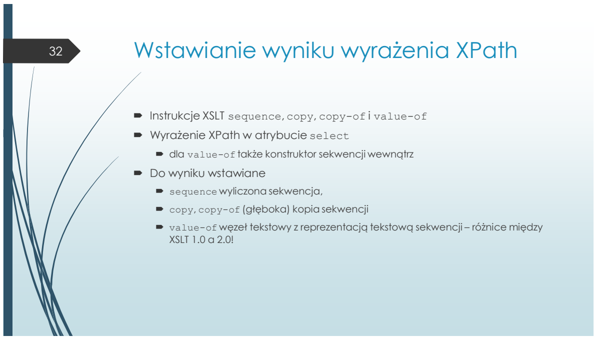
31

Nazwane zbiory atrybutów

- `xsl:attribute-set`
  - Jest elementem pierwszego poziomu
  - Grupuje atrybuty w jedną całość
- `xsl:attribute`
  - Wartość atrybutu jest przechowywana w elemencie o atrybucie `name`

```
<xsl:attribute-set name="atrybuty">
  <xsl:attribute name="style">
        color:white;background-color:black; font-size:14pt
  </xsl:attribute>
  <xsl:attribute name="height">50</xsl:attribute>
  <xsl:attribute name="width">50</xsl:attribute>
  <xsl:attribute name="align">center</xsl:attribute>
</xsl:attribute-set>
```

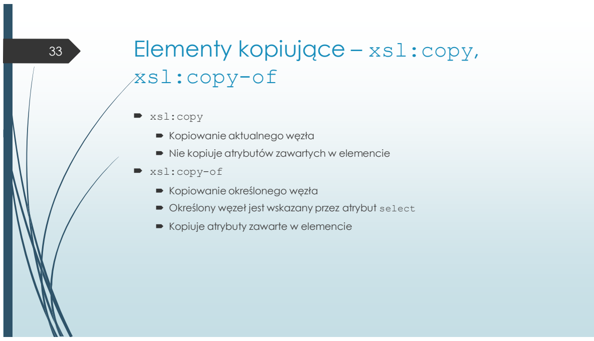
```
<table xsl:use-attribute-sets="atrybuty">
  ...
</table>
```



32

Wstawianie wyniku wyrażenia XPath

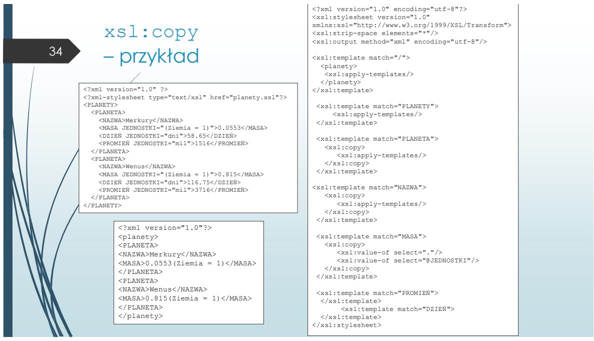
- Instrukcje XSLT `sequence`, `copy`, `copy-of` i `value-of`
- Wyrażenie XPath w atrybucie `select`
  - dla `value-of` także konstruktor sekwencji wewnątrz
- Do wyniku wstawiane
  - `sequence` wyliczona sekwencja,
  - `copy`, `copy-of` (głęboka) kopia sekwencji
  - `value-of` węzeł tekstowy z reprezentacją tekstową sekwencji – różnice między XSLT 1.0 a 2.0!



33

Elementy kopiujące – `xsl:copy`, `xsl:copy-of`

- `xsl:copy`
  - Kopiowanie aktualnego węzła
  - Nie kopiuje atrybutów zawartych w elemencie
- `xsl:copy-of`
  - Kopiowanie określonego węzła
  - Określony węzeł jest wskazany przez atrybut `select`
  - Kopiuje atrybuty zawarte w elemencie



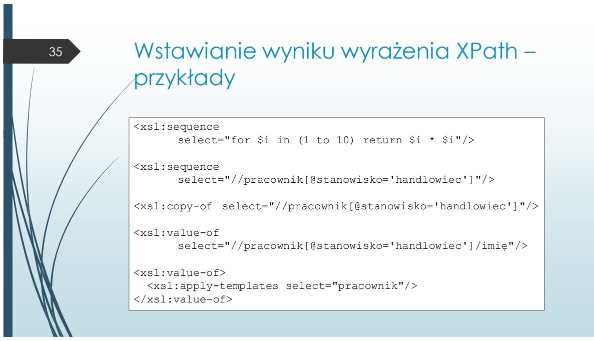
34

`xsl:copy` – przykład

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="planety.xsl"?>
<PLANETY>
  <PLANETA>
    <NAZWA>Merkury</NAZWA>
    <MASA JEDNOSTKI="(Ziemia = 1)">0.0553</MASA>
    <DZIEŃ JEDNOSTKI="dni">58.65</DZIEŃ>
    <PROMIEŃ JEDNOSTKI="mil">1516</PROMIEŃ>
  </PLANETA>
  <PLANETA>
    <NAZWA>Wenus</NAZWA>
    <MASA JEDNOSTKI="(Ziemia = 1)">0.815</MASA>
    <DZIEŃ JEDNOSTKI="dni">116.75</DZIEŃ>
    <PROMIEŃ JEDNOSTKI="mil">3716</PROMIEŃ>
  </PLANETA>
</PLANETY>
```

```
<?xml version="1.0"?>
<planety>
<PLANETA>
<NAZWA>Merkury</NAZWA>
<MASA>0.0553(Ziemia = 1)</MASA>
</PLANETA>
<PLANETA>
<NAZWA>Wenus</NAZWA>
<MASA>0.815(Ziemia = 1)</MASA>
</PLANETA>
</planety>
```

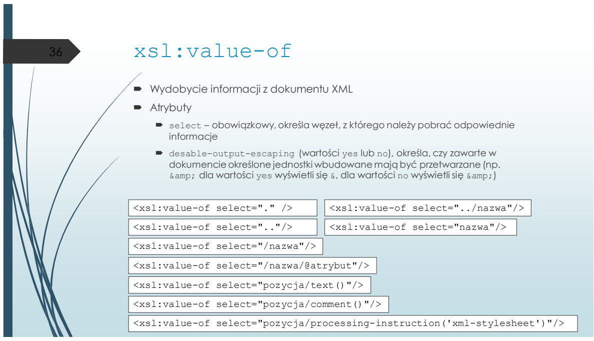
```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:strip-space elements="*"/>
<xsl:output method="xml" encoding="utf-8"/>

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

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

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

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

 <xsl:template match="MASA">
   <xsl:copy>
      <xsl:value-of select="."/>
      <xsl:value-of select="@JEDNOSTKI"/>
   </xsl:copy>
 </xsl:template>

 <xsl:template match="PROMIEŃ">
  </xsl:template>
       <xsl:template match="DZIEŃ">
  </xsl:template>
</xsl:stylesheet>
```



35

Wstawianie wyniku wyrażenia XPath – przykłady

```
<xsl:sequence
       select="for $i in (1 to 10) return $i * $i"/>

<xsl:sequence
       select="//pracownik[@stanowisko='handlowiec']"/>

<xsl:copy-of  select="//pracownik[@stanowisko='handlowiec']"/>

<xsl:value-of
       select="//pracownik[@stanowisko='handlowiec']/imię"/>

<xsl:value-of>
  <xsl:apply-templates select="pracownik"/>
</xsl:value-of>
```



36

`xsl:value-of`

- Wydobycie informacji z dokumentu XML
- Atrybuty
  - `select` – obowiązkowy, określa węzeł, z którego należy pobrać odpowiednie informacje
  - `desable-output-escaping` (wartości `yes` lub `no`), określa, czy zawarte w dokumencie określone jednostki wbudowane mają być przetwarzane (np. `&amp;` dla wartości `yes` wyświetli się `&`, dla wartości `no` wyświetli się `&amp;`)

```
<xsl:value-of select="." />
<xsl:value-of select=".."/>
<xsl:value-of select="/nazwa"/>
<xsl:value-of select="/nazwa/@atrybut"/>
<xsl:value-of select="pozycja/text()"/>
<xsl:value-of select="pozycja/comment()"/>
<xsl:value-of select="pozycja/processing-instruction('xml-stylesheet')"/>
```

```
<xsl:value-of select="../nazwa"/>
<xsl:value-of select="nazwa"/>
```

37

xsl:value-of – wykorzystanie wieloznacznika

```
<xsl:template match="pozycja">
       <xsl:value-of select="*"/>
</xsl:template>
```

```
<xsl:template match="pozycja">
       <xsl:value-of select="@*"/>
</xsl:template>
```

```
<xsl:template match="pozycja">
       <xsl:value-of select="* | @*"/>
</xsl:template>
```

```
<xsl:template match="pozycja">
       <xsl:value-of select="node()"/>
</xsl:template>
```

38

xsl:value-of w XSLT 1.0

- Jeśli wartość do wypisania jest wieloelementowym zbiorem węzłów, to do wyniku przekształcenia przepisywane jest rzutowanie na `string` tylko pierwszego węzła ze zbioru

Dokument

```
<osoba><imię>Jan</imię><nazwisko>Kowalski</nazwisko></osoba>
<osoba><imię>Zofia</imię><nazwisko>Nowak</nazwisko></osoba>
```

Arkusz

```
<wynik><xsl:value-of select="//osoba/imię"/></wynik>
```

Arkusz

```
<wynik>Jan</wynik>
```

39

xsl:value-of w XSLT 2.0

- Sekwencja poddana atomizacji
- Rzutowanie każdego atomu na `string`
- Wypisane rozdzielone spacjami
- Możliwość podania własnego separatora (atrybut `separator`)

Dokument

```
<osoba><imię>Jan</imię><nazwisko>Kowalski</nazwisko></osoba>
<osoba><imię>Zofia</imię><nazwisko>Nowak</nazwisko></osoba>
```

Arkusz

```
<wynik><xsl:value-of select="//osoba/imię"/></wynik>
```

Arkusz

```
<wynik>Jan Zofia</wynik>
```

40

Szablony wartości atrybutów

- Wygodny sposób na dynamiczne obliczenie wartości atrybutu
- Można używać w
  - atrybutach wstawianych do wyniku
  - niektórych atrybutach instrukcji XSLT
- Części stałe – po prostu napisy kopiowane do wyniku
  - { i } zapisywane jako {{ i }}
- Części zmienne – obliczane dynamicznie
  - wyrażenie XPath umieszczone między { a }
  - wstawiana reprezentacja tekstowa wyliczonej sekwencji (jak w `value-of`, ze spacją jako separatorem)
  - także analogiczne różnice między XSLT 1.0 a XSLT 2.0

```
<img src="{$image_server}/{@id}.jpg"/>

<xsl:element name="h{count(ancestror-or-self::sekcja)}">
  ...
</xsl:element>
```

41

Funkcje sterujące w XSLT

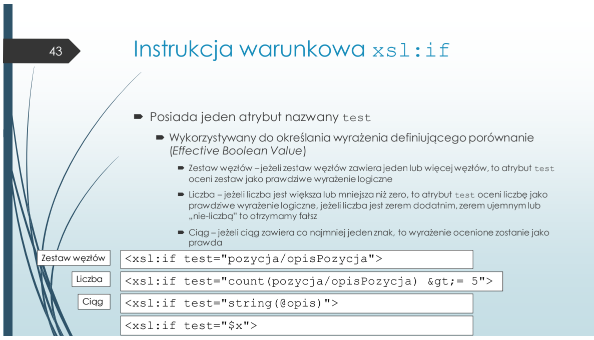
- Służą do warunkowego przetwarzania węzłów
  - `xsl:if`
    - co należy zrobić, jeśli wartość jest zgodna z wyrażeniem (if-then)
  - `xsl:choose, xsl:otherwise`
    - co należy zrobić, jeśli wartość jest zgodna z wyrażeniem oraz co należy zrobić jeśli wartość nie odpowiada wyrażeniu (if-then-else)
  - `xsl:choose, xsl:when`
    - określa zestaw wartości, które powinny być porównane oraz określa, co powinno się zdarzyć, kiedy zgodności te mają miejsce, można też zdefiniować co powinno się zdarzyć w sytuacji braku zgodności (switch-case)
  - `xsl:for-each`
    - określa sposób przetwarzania każdej wartości danego zestawu wartości, przetwarzanie iteracyjne (for-each)

42

Instrukcja warunkowa xsl:if

- Polecenie `if` może występować w dowolnym miejscu definiowania szablonu
- Posiada jeden atrybut nazwany `test`
  - Wykorzystywany do określania wyrażenia definiującego porównanie (*Effective Boolean Value*)
    - Zestaw węzłów – jeżeli zestaw węzłów zawiera jeden lub więcej węzłów, to atrybut `test` oceni zestaw jako prawdziwe wyrażenie logiczne
    - Liczba – jeżeli liczba jest większa lub mniejsza niż zero, to atrybut `test` oceni liczbę jako prawdziwe wyrażenie logiczne, jeżeli liczba jest zerem dodatnim, zerem ujemnym lub „nie-liczbą" to otrzymamy fałsz
    - Ciąg – jeżeli ciąg zawiera co najmniej jeden znak, to wyrażenie ocenione zostanie jako prawda
  - Wartością jest wyrażenie XPath uzupełnione funkcjami standardu XSLT

```
<xsl:if test="wyrażenieXPath"> … </xsl:if>
```

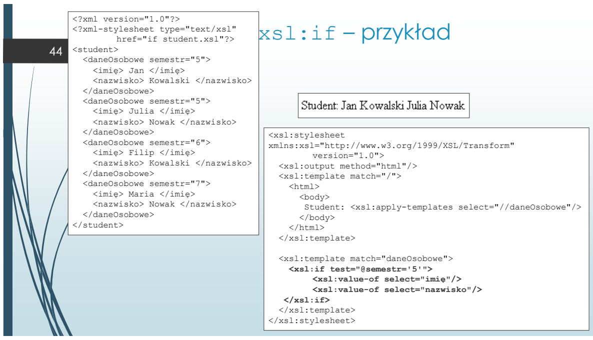


43

Instrukcja warunkowa `xsl:if`

- Posiada jeden atrybut nazwany `test`
  - Wykorzystywany do określania wyrażenia definiującego porównanie (*Effective Boolean Value*)
    - Zestaw węzłów – jeżeli zestaw węzłów zawiera jeden lub więcej węzłów, to atrybut `test` oceni zestaw jako prawdziwe wyrażenie logiczne
    - Liczba – jeżeli liczba jest większa lub mniejsza niż zero, to atrybut `test` oceni liczbę jako prawdziwe wyrażenie logiczne, jeżeli liczba jest zerem dodatnim, zerem ujemnym lub „nie-liczbą" to otrzymamy fałsz
    - Ciąg – jeżeli ciąg zawiera co najmniej jeden znak, to wyrażenie ocenione zostanie jako prawda

Zestaw węzłów
```
<xsl:if test="pozycja/opisPozycja">
```
Liczba
```
<xsl:if test="count(pozycja/opisPozycja) &gt;= 5">
```
Ciąg
```
<xsl:if test="string(@opis)">
```
```
<xsl:if test="$x">
```



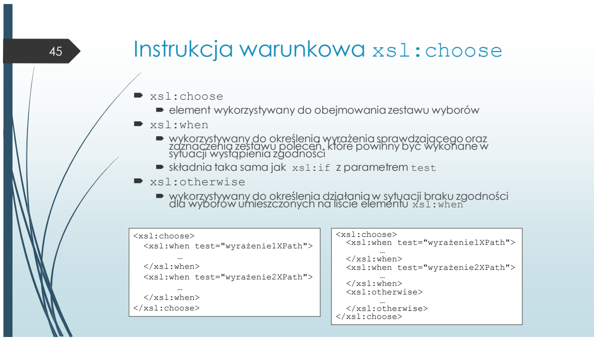
44

`xsl:if` – przykład

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
        href="if student.xsl"?>
<student>
  <daneOsobowe semestr="5">
    <imię> Jan </imię>
    <nazwisko> Kowalski </nazwisko>
  </daneOsobowe>
  <daneOsobowe semestr="5">
    <imię> Julia </imię>
    <nazwisko> Nowak </nazwisko>
  </daneOsobowe>
  <daneOsobowe semestr="6">
    <imię> Filip </imię>
    <nazwisko> Kowalski </nazwisko>
  </daneOsobowe>
  <daneOsobowe semestr="7">
    <imię> Maria </imię>
    <nazwisko> Nowak </nazwisko>
  </daneOsobowe>
</student>
```

Student: Jan Kowalski Julia Nowak

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <body>
       Student: <xsl:apply-templates select="//daneOsobowe"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="daneOsobowe">
    <xsl:if test="@semestr='5'">
         <xsl:value-of select="imię"/>
         <xsl:value-of select="nazwisko"/>
   </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```



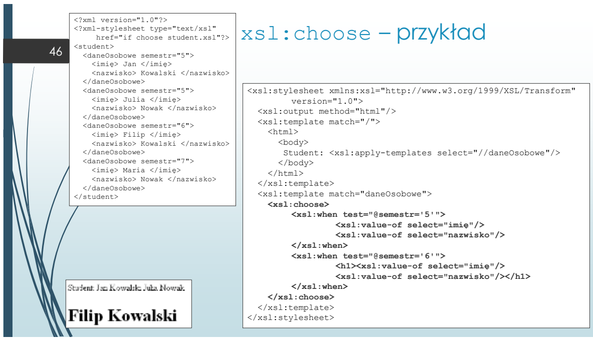
45

Instrukcja warunkowa `xsl:choose`

- `xsl:choose`
  - element wykorzystywany do obejmowania zestawu wyborów
- `xsl:when`
  - wykorzystywany do określenia wyrażenia sprawdzającego oraz zaznaczenia zestawu poleceń, które powinny być wykonane w sytuacji wystąpienia zgodności
  - składnia taka sama jak `xsl:if` z parametrem `test`
- `xsl:otherwise`
  - wykorzystywany do określenia działania w sytuacji braku zgodności dla wyborów umieszczonych na liście elementu `xsl:when`

```
<xsl:choose>
  <xsl:when test="wyrażenie1XPath">
        …
  </xsl:when>
  <xsl:when test="wyrażenie2XPath">
        …
  </xsl:when>
</xsl:choose>
```

```
<xsl:choose>
  <xsl:when test="wyrażenie1XPath">
        …
  </xsl:when>
  <xsl:when test="wyrażenie2XPath">
        …
  </xsl:when>
  <xsl:otherwise>
        …
  </xsl:otherwise>
</xsl:choose>
```



46

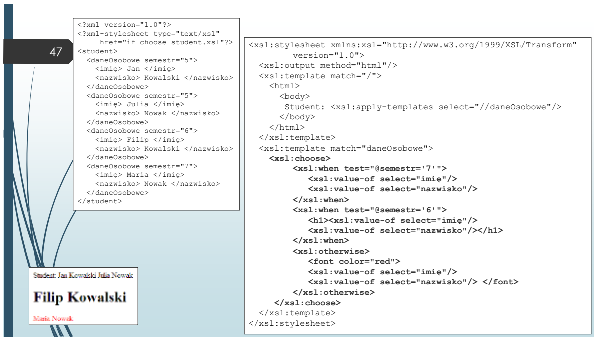
`xsl:choose` – przykład

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
     href="if choose student.xsl"?>
<student>
  <daneOsobowe semestr="5">
    <imię> Jan </imię>
    <nazwisko> Kowalski </nazwisko>
  </daneOsobowe>
  <daneOsobowe semestr="5">
    <imię> Julia </imię>
    <nazwisko> Nowak </nazwisko>
  </daneOsobowe>
  <daneOsobowe semestr="6">
    <imię> Filip </imię>
    <nazwisko> Kowalski </nazwisko>
  </daneOsobowe>
  <daneOsobowe semestr="7">
    <imię> Maria </imię>
    <nazwisko> Nowak </nazwisko>
  </daneOsobowe>
</student>
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <body>
       Student: <xsl:apply-templates select="//daneOsobowe"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="daneOsobowe">
    <xsl:choose>
        <xsl:when test="@semestr='5'">
                <xsl:value-of select="imię"/>
                <xsl:value-of select="nazwisko"/>
        </xsl:when>
        <xsl:when test="@semestr='6'">
                <h1><xsl:value-of select="imię"/>
                <xsl:value-of select="nazwisko"/></h1>
        </xsl:when>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

Student: Jan Kowalski Julia Nowak

Filip Kowalski



47

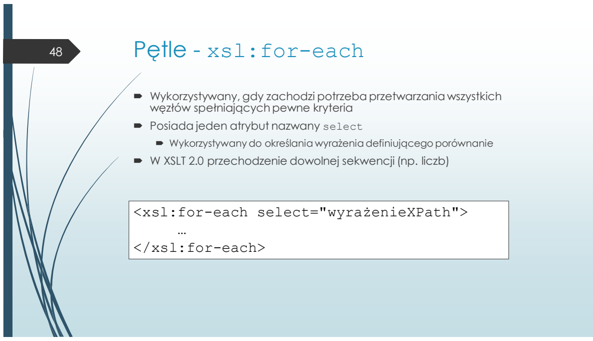
```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
     href="if choose student.xsl"?>
<student>
  <daneOsobowe semestr="5">
    <imię> Jan </imię>
    <nazwisko> Kowalski </nazwisko>
  </daneOsobowe>
  <daneOsobowe semestr="5">
    <imię> Julia </imię>
    <nazwisko> Nowak </nazwisko>
  </daneOsobowe>
  <daneOsobowe semestr="6">
    <imię> Filip </imię>
    <nazwisko> Kowalski </nazwisko>
  </daneOsobowe>
  <daneOsobowe semestr="7">
    <imię> Maria </imię>
    <nazwisko> Nowak </nazwisko>
  </daneOsobowe>
</student>
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <body>
       Student: <xsl:apply-templates select="//daneOsobowe"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="daneOsobowe">
    <xsl:choose>
        <xsl:when test="@semestr='7'">
           <xsl:value-of select="imię"/>
           <xsl:value-of select="nazwisko"/>
        </xsl:when>
        <xsl:when test="@semestr='6'">
           <h1><xsl:value-of select="imię"/>
           <xsl:value-of select="nazwisko"/></h1>
        </xsl:when>
        <xsl:otherwise>
           <font color="red">
           <xsl:value-of select="imię"/>
           <xsl:value-of select="nazwisko"/> </font>
        </xsl:otherwise>
     </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

Student: Jan Kowalski Julia Nowak

Filip Kowalski

Maria Nowak

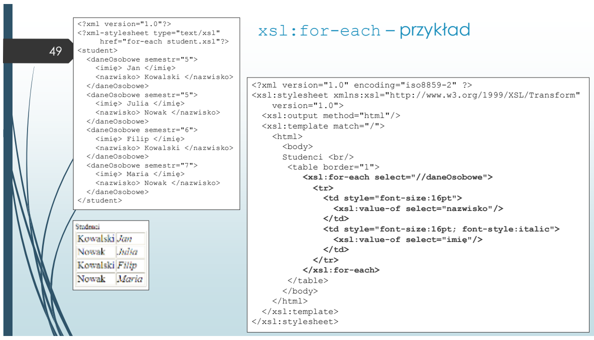


48

Pętle - `xsl:for-each`

- Wykorzystywany, gdy zachodzi potrzeba przetwarzania wszystkich węzłów spełniających pewne kryteria
- Posiada jeden atrybut nazwany `select`
  - Wykorzystywany do określania wyrażenia definiującego porównanie
- W XSLT 2.0 przechodzenie dowolnej sekwencji (np. liczb)

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



49

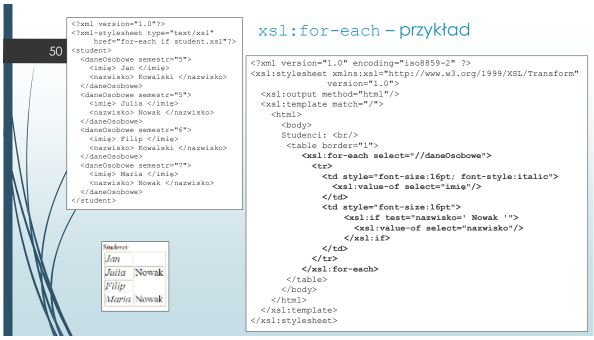
xsl:for-each – przykład

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
     href="for-each student.xsl"?>
<student>
  <daneOsobowe semestr="5">
    <imię> Jan </imię>
    <nazwisko> Kowalski </nazwisko>
  </daneOsobowe>
  <daneOsobowe semestr="5">
    <imię> Julia </imię>
    <nazwisko> Nowak </nazwisko>
  </daneOsobowe>
  <daneOsobowe semestr="6">
    <imię> Filip </imię>
    <nazwisko> Kowalski </nazwisko>
  </daneOsobowe>
  <daneOsobowe semestr="7">
    <imię> Maria </imię>
    <nazwisko> Nowak </nazwisko>
  </daneOsobowe>
</student>
```

```
<?xml version="1.0" encoding="iso8859-2" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <body>
      Studenci <br/>
       <table border="1">
          <xsl:for-each select="//daneOsobowe">
            <tr>
              <td style="font-size:16pt">
                <xsl:value-of select="nazwisko"/>
              </td>
              <td style="font-size:16pt; font-style:italic">
                <xsl:value-of select="imię"/>
              </td>
            </tr>
          </xsl:for-each>
       </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Studenci

Kowalski	*Jan*
Nowak	*Julia*
Kowalski	*Filip*
Nowak	*Maria*



50

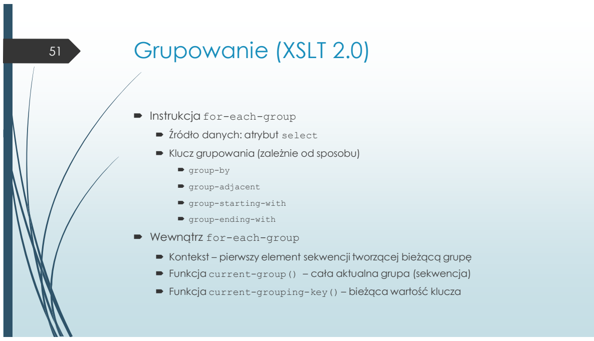
xsl:for-each – przykład

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
     href="for-each if student.xsl"?>
<student>
  <daneOsobowe semestr="5">
    <imię> Jan </imię>
    <nazwisko> Kowalski </nazwisko>
  </daneOsobowe>
  <daneOsobowe semestr="5">
    <imię> Julia </imię>
    <nazwisko> Nowak </nazwisko>
  </daneOsobowe>
  <daneOsobowe semestr="6">
    <imię> Filip </imię>
    <nazwisko> Kowalski </nazwisko>
  </daneOsobowe>
  <daneOsobowe semestr="7">
    <imię> Maria </imię>
    <nazwisko> Nowak </nazwisko>
  </daneOsobowe>
</student>
```

```
<?xml version="1.0" encoding="iso8859-2" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
               version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <body>
      Studenci: <br/>
       <table border="1">
          <xsl:for-each select="//daneOsobowe">
            <tr>
              <td style="font-size:16pt; font-style:italic">
                <xsl:value-of select="imię"/>
              </td>
              <td style="font-size:16pt">
                  <xsl:if test="nazwisko=' Nowak '">
                    <xsl:value-of select="nazwisko"/>
                  </xsl:if>
              </td>
            </tr>
          </xsl:for-each>
       </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Studenci:

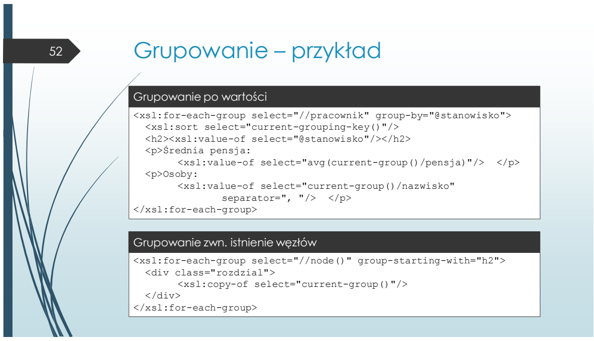
Jan	
Julia	Nowak
Filip	
Maria	Nowak



51

Grupowanie (XSLT 2.0)

- Instrukcja `for-each-group`
  - Źródło danych: atrybut `select`
  - Klucz grupowania (zależnie od sposobu)
    - `group-by`
    - `group-adjacent`
    - `group-starting-with`
    - `group-ending-with`
- Wewnątrz `for-each-group`
  - Kontekst – pierwszy element sekwencji tworzącej bieżącą grupę
  - Funkcja `current-group()` – cała aktualna grupa (sekwencja)
  - Funkcja `current-grouping-key()` – bieżąca wartość klucza



52

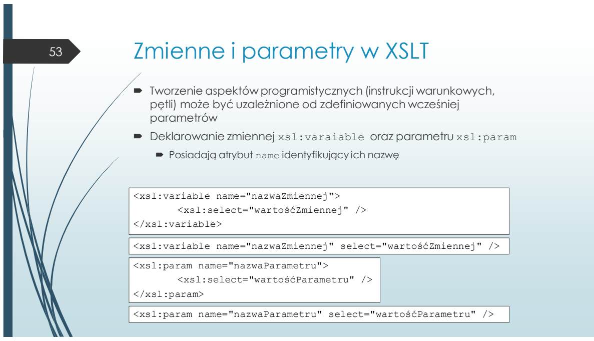
Grupowanie – przykład

Grupowanie po wartości

```
<xsl:for-each-group select="//pracownik" group-by="@stanowisko">
  <xsl:sort select="current-grouping-key()"/>
  <h2><xsl:value-of select="@stanowisko"/></h2>
  <p>Średnia pensja:
        <xsl:value-of select="avg(current-group()/pensja)"/>  </p>
  <p>Osoby:
        <xsl:value-of select="current-group()/nazwisko"
               separator=", "/>  </p>
</xsl:for-each-group>
```

Grupowanie zwn. istnienie węzłów

```
<xsl:for-each-group select="//node()" group-starting-with="h2">
  <div class="rozdzial">
        <xsl:copy-of select="current-group()"/>
  </div>
</xsl:for-each-group>
```



53

Zmienne i parametry w XSLT

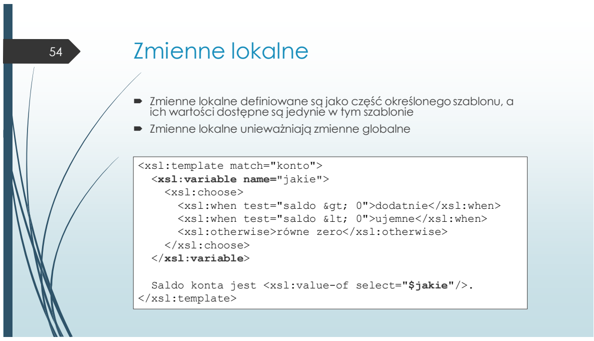
- Tworzenie aspektów programistycznych (instrukcji warunkowych, pętli) może być uzależnione od zdefiniowanych wcześniej parametrów
- Deklarowanie zmiennej `xsl:varaiable` oraz parametru `xsl:param`
  - Posiadają atrybut `name` identyfikujący ich nazwę

```
<xsl:variable name="nazwaZmiennej">
       <xsl:select="wartośćZmiennej" />
</xsl:variable>
```

```
<xsl:variable name="nazwaZmiennej" select="wartośćZmiennej" />
```

```
<xsl:param name="nazwaParametru">
       <xsl:select="wartośćParametru" />
</xsl:param>
```

```
<xsl:param name="nazwaParametru" select="wartośćParametru" />
```

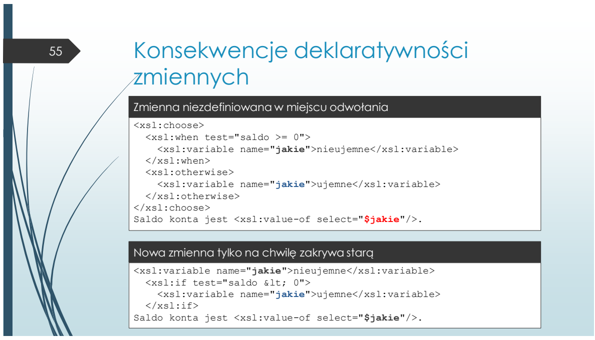


54

Zmienne lokalne

- Zmienne lokalne definiowane są jako część określonego szablonu, a ich wartości dostępne są jedynie w tym szablonie
- Zmienne lokalne unieważniają zmienne globalne

```
<xsl:template match="konto">
  <xsl:variable name="jakie">
    <xsl:choose>
      <xsl:when test="saldo &gt; 0">dodatnie</xsl:when>
      <xsl:when test="saldo &lt; 0">ujemne</xsl:when>
      <xsl:otherwise>równe zero</xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  Saldo konta jest <xsl:value-of select="$jakie"/>.
</xsl:template>
```



55

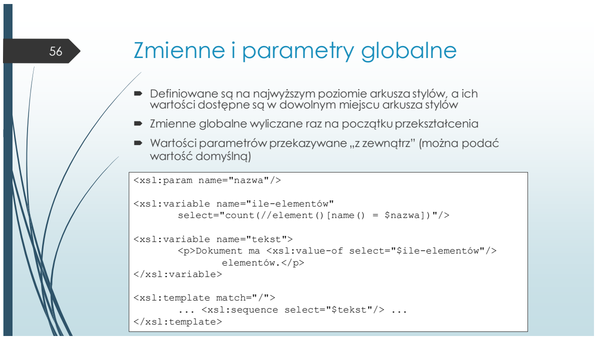
Konsekwencje deklaratywności zmiennych

Zmienna niezdefiniowana w miejscu odwołania

```
<xsl:choose>
  <xsl:when test="saldo >= 0">
    <xsl:variable name="jakie">nieujemne</xsl:variable>
  </xsl:when>
  <xsl:otherwise>
    <xsl:variable name="jakie">ujemne</xsl:variable>
  </xsl:otherwise>
</xsl:choose>
Saldo konta jest <xsl:value-of select="$jakie"/>.
```

Nowa zmienna tylko na chwilę zakrywa starą

```
<xsl:variable name="jakie">nieujemne</xsl:variable>
  <xsl:if test="saldo &lt; 0">
    <xsl:variable name="jakie">ujemne</xsl:variable>
  </xsl:if>
Saldo konta jest <xsl:value-of select="$jakie"/>.
```

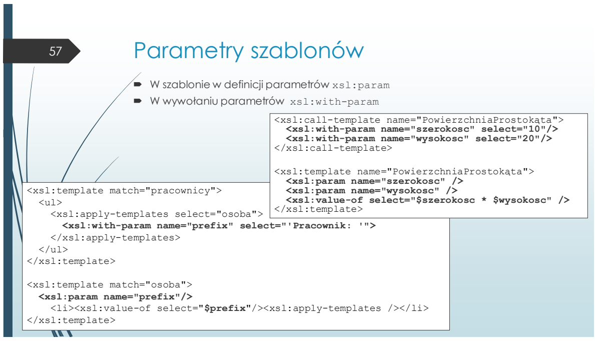


56

Zmienne i parametry globalne

- Definiowane są na najwyższym poziomie arkusza stylów, a ich wartości dostępne są w dowolnym miejscu arkusza stylów
- Zmienne globalne wyliczane raz na początku przekształcenia
- Wartości parametrów przekazywane „z zewnątrz" (można podać wartość domyślną)

```
<xsl:param name="nazwa"/>

<xsl:variable name="ile-elementów"
       select="count(//element()[name() = $nazwa])"/>

<xsl:variable name="tekst">
       <p>Dokument ma <xsl:value-of select="$ile-elementów"/>
              elementów.</p>
</xsl:variable>

<xsl:template match="/">
       ... <xsl:sequence select="$tekst"/> ...
</xsl:template>
```



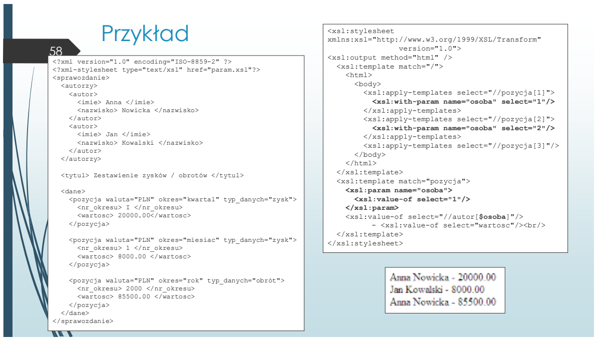
57

Parametry szablonów

- W szablonie w definicji parametrów `xsl:param`
- W wywołaniu parametrów `xsl:with-param`

```
<xsl:call-template name="PowierzchniaProstokąta">
  <xsl:with-param name="szerokosc" select="10"/>
  <xsl:with-param name="wysokosc" select="20"/>
</xsl:call-template>

<xsl:template name="PowierzchniaProstokąta">
  <xsl:param name="szerokosc" />
  <xsl:param name="wysokosc" />
  <xsl:value-of select="$szerokosc * $wysokosc" />
</xsl:template>
```

```
<xsl:template match="pracownicy">
  <ul>
    <xsl:apply-templates select="osoba">
      <xsl:with-param name="prefix" select="'Pracownik: '">
    </xsl:apply-templates>
  </ul>
</xsl:template>

<xsl:template match="osoba">
  <xsl:param name="prefix"/>
    <li><xsl:value-of select="$prefix"/><xsl:apply-templates /></li>
</xsl:template>
```



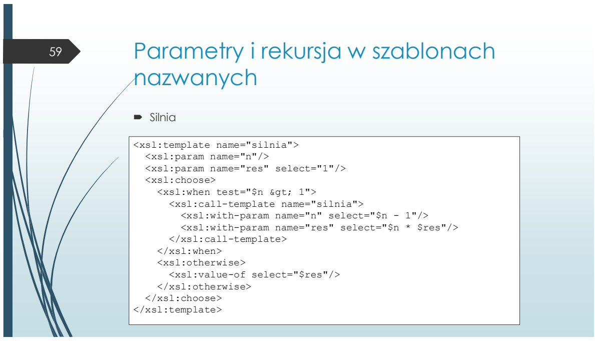
58

Przykład

```
<?xml version="1.0" encoding="ISO-8859-2" ?>
<?xml-stylesheet type="text/xsl" href="param.xsl"?>
<sprawozdanie>
  <autorzy>
    <autor>
      <imie> Anna </imie>
      <nazwisko> Nowicka </nazwisko>
    </autor>
    <autor>
      <imie> Jan </imie>
      <nazwisko> Kowalski </nazwisko>
    </autor>
  </autorzy>

  <tytul> Zestawienie zysków / obrotów </tytul>

  <dane>
    <pozycja waluta="PLN" okres="kwartal" typ_danych="zysk">
      <nr_okresu> I </nr_okresu>
      <wartosc> 20000.00</wartosc>
    </pozycja>

    <pozycja waluta="PLN" okres="miesiac" typ_danych="zysk">
      <nr_okresu> 1 </nr_okresu>
      <wartosc> 8000.00 </wartosc>
    </pozycja>

    <pozycja waluta="PLN" okres="rok" typ_danych="obrót">
      <nr_okresu> 2000 </nr_okresu>
      <wartosc> 85500.00 </wartosc>
    </pozycja>
  </dane>
</sprawozdanie>
```

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
<xsl:output method="html" />
  <xsl:template match="/">
    <html>
      <body>
        <xsl:apply-templates select="//pozycja[1]">
          <xsl:with-param name="osoba" select="1"/>
        </xsl:apply-templates>
        <xsl:apply-templates select="//pozycja[2]">
          <xsl:with-param name="osoba" select="2"/>
        </xsl:apply-templates>
        <xsl:apply-templates select="//pozycja[3]"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="pozycja">
    <xsl:param name="osoba">
      <xsl:value-of select="1"/>
    </xsl:param>
    <xsl:value-of select="//autor[$osoba]"/>
          - <xsl:value-of select="wartosc"/><br/>
  </xsl:template>
</xsl:stylesheet>
```

```
Anna Nowicka - 20000.00
Jan Kowalski - 8000.00
Anna Nowicka - 85500.00
```

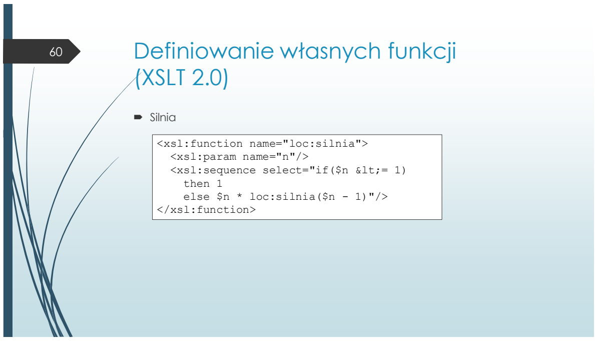


59

Parametry i rekursja w szablonach nazwanych

- Silnia

```
<xsl:template name="silnia">
  <xsl:param name="n"/>
  <xsl:param name="res" select="1"/>
  <xsl:choose>
    <xsl:when test="$n &gt; 1">
      <xsl:call-template name="silnia">
        <xsl:with-param name="n" select="$n - 1"/>
        <xsl:with-param name="res" select="$n * $res"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$res"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

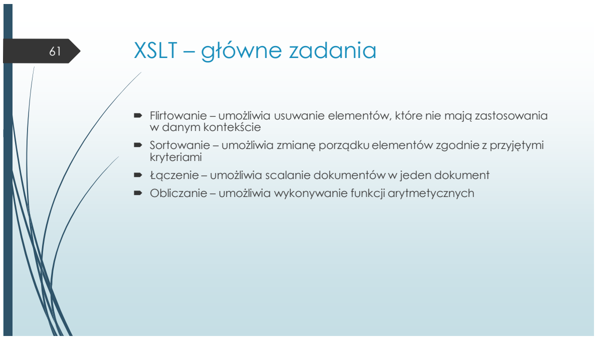


60

Definiowanie własnych funkcji (XSLT 2.0)

- Silnia

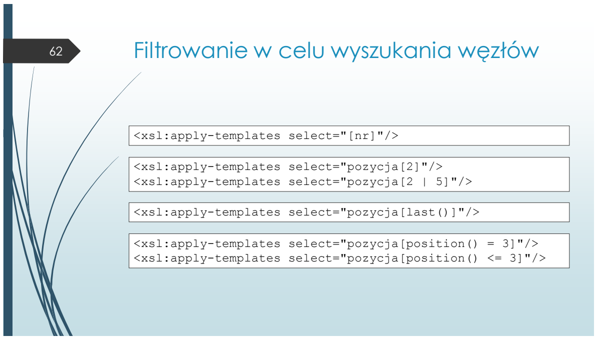
```
<xsl:function name="loc:silnia">
  <xsl:param name="n"/>
  <xsl:sequence select="if($n &lt;= 1)
   then 1
   else $n * loc:silnia($n - 1)"/>
</xsl:function>
```



61

XSLT – główne zadania

- Flirtowanie – umożliwia usuwanie elementów, które nie mają zastosowania w danym kontekście
- Sortowanie – umożliwia zmianę porządku elementów zgodnie z przyjętymi kryteriami
- Łączenie – umożliwia scalanie dokumentów w jeden dokument
- Obliczanie – umożliwia wykonywanie funkcji arytmetycznych



62

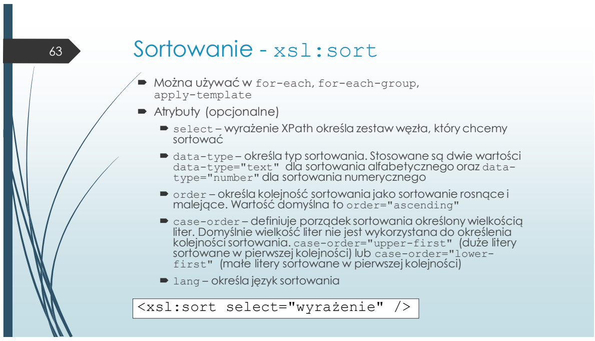
Filtrowanie w celu wyszukania węzłów

```
<xsl:apply-templates select="[nr]"/>
```

```
<xsl:apply-templates select="pozycja[2]"/>
<xsl:apply-templates select="pozycja[2 | 5]"/>
```

```
<xsl:apply-templates select="pozycja[last()]"/>
```

```
<xsl:apply-templates select="pozycja[position() = 3]"/>
<xsl:apply-templates select="pozycja[position() <= 3]"/>
```

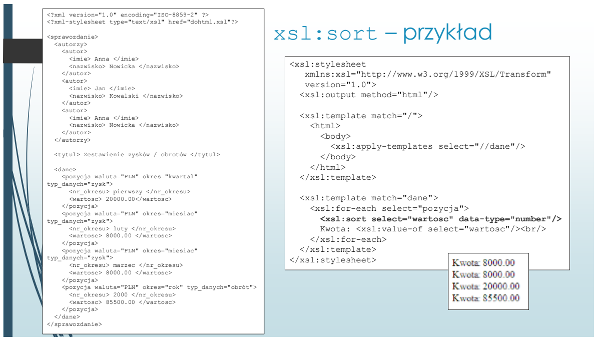


63

Sortowanie - xsl:sort

- Można używać w for-each, for-each-group, apply-template
- Atrybuty (opcjonalne)
  - select – wyrażenie XPath określa zestaw węzła, który chcemy sortować
  - data-type – określa typ sortowania. Stosowane są dwie wartości data-type="text" dla sortowania alfabetycznego oraz data-type="number" dla sortowania numerycznego
  - order – określa kolejność sortowania jako sortowanie rosnące i malejące. Wartość domyślna to order="ascending"
  - case-order – definiuje porządek sortowania określony wielkością liter. Domyślnie wielkość liter nie jest wykorzystana do określenia kolejności sortowania. case-order="upper-first" (duże litery sortowane w pierwszej kolejności) lub case-order="lower-first" (małe litery sortowane w pierwszej kolejności)
  - lang – określa język sortowania

```
<xsl:sort select="wyrażenie" />
```

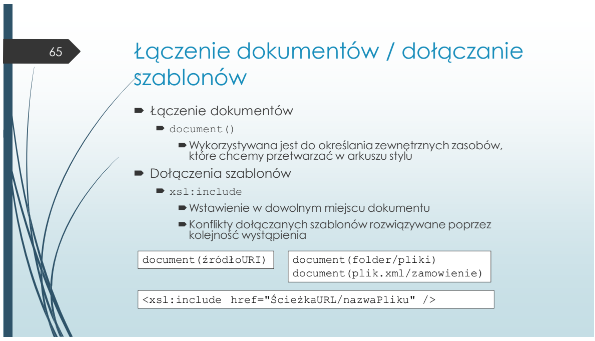


```
<?xml version="1.0" encoding="ISO-8859-2" ?>
<?xml-stylesheet type="text/xsl" href="dohtml.xsl"?>

<sprawozdanie>
  <autorzy>
    <autor>
      <imie> Anna </imie>
      <nazwisko> Nowicka </nazwisko>
    </autor>
    <autor>
      <imie> Jan </imie>
      <nazwisko> Kowalski </nazwisko>
    </autor>
    <autor>
      <imie> Anna </imie>
      <nazwisko> Nowicka </nazwisko>
    </autor>
  </autorzy>

  <tytul> Zestawienie zysków / obrotów </tytul>

  <dane>
    <pozycja waluta="PLN" okres="kwartal"
typ_danych="zysk">
      <nr_okresu> pierwszy </nr_okresu>
      <wartosc> 20000.00</wartosc>
    </pozycja>
    <pozycja waluta="PLN" okres="miesiac"
typ_danych="zysk">
      <nr_okresu> luty </nr_okresu>
      <wartosc> 8000.00 </wartosc>
    </pozycja>
    <pozycja waluta="PLN" okres="miesiac"
typ_danych="zysk">
      <nr_okresu> marzec </nr_okresu>
      <wartosc> 8000.00 </wartosc>
    </pozycja>
    <pozycja waluta="PLN" okres="rok" typ_danych="obrót">
      <nr_okresu> 2000 </nr_okresu>
      <wartosc> 85500.00 </wartosc>
    </pozycja>
  </dane>
</sprawozdanie>
```

xsl:sort – przykład

```
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <body>
        <xsl:apply-templates select="//dane"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="dane">
    <xsl:for-each select="pozycja">
      <xsl:sort select="wartosc" data-type="number"/>
      Kwota: <xsl:value-of select="wartosc"/><br/>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

```
Kwota: 8000.00
Kwota: 8000.00
Kwota: 20000.00
Kwota: 85500.00
```



65

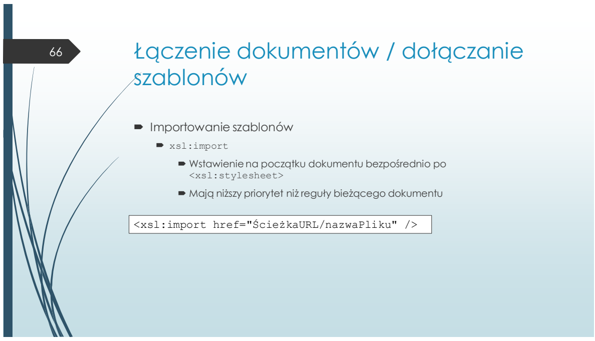
Łączenie dokumentów / dołączanie szablonów

- Łączenie dokumentów
  - document()
    - Wykorzystywana jest do określania zewnętrznych zasobów, które chcemy przetwarzać w arkuszu stylu
- Dołączenia szablonów
  - xsl:include
    - Wstawienie w dowolnym miejscu dokumentu
    - Konflikty dołączanych szablonów rozwiązywane poprzez kolejność wystąpienia

```
document(źródłoURI)
```

```
document(folder/pliki)
document(plik.xml/zamowienie)
```

```
<xsl:include  href="ŚcieżkaURL/nazwaPliku" />
```

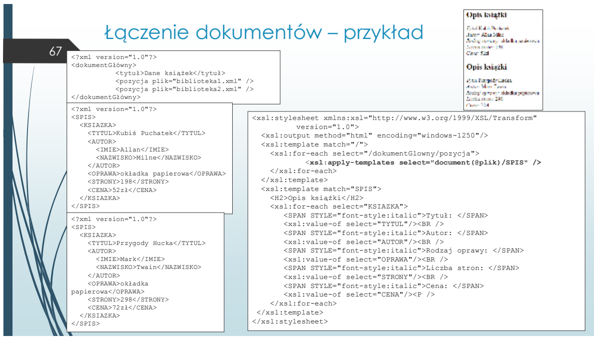


66

Łączenie dokumentów / dołączanie szablonów

- Importowanie szablonów
  - xsl:import
    - Wstawienie na początku dokumentu bezpośrednio po <xsl:stylesheet>
    - Mają niższy priorytet niż reguły bieżącego dokumentu

```
<xsl:import href="ŚcieżkaURL/nazwaPliku" />
```



67

Łączenie dokumentów – przykład

```
<?xml version="1.0"?>
<dokumentGłówny>
          <tytuł>Dane książek</tytuł>
          <pozycja plik="biblioteka1.xml" />
          <pozycja plik="biblioteka2.xml" />
</dokumentGłówny>
```

```
<?xml version="1.0"?>
<SPIS>
  <KSIAZKA>
    <TYTUL>Kubiś Puchatek</TYTUL>
    <AUTOR>
      <IMIE>Allan</IMIE>
      <NAZWISKO>Milne</NAZWISKO>
    </AUTOR>
    <OPRAWA>okładka papierowa</OPRAWA>
    <STRONY>198</STRONY>
    <CENA>52zł</CENA>
  </KSIAZKA>
</SPIS>
```

```
<?xml version="1.0"?>
<SPIS>
  <KSIAZKA>
    <TYTUL>Przygody Hucka</TYTUL>
    <AUTOR>
      <IMIE>Mark</IMIE>
      <NAZWISKO>Twain</NAZWISKO>
    </AUTOR>
    <OPRAWA>okładka
papierowa</OPRAWA>
    <STRONY>298</STRONY>
    <CENA>72zł</CENA>
  </KSIAZKA>
</SPIS>
```

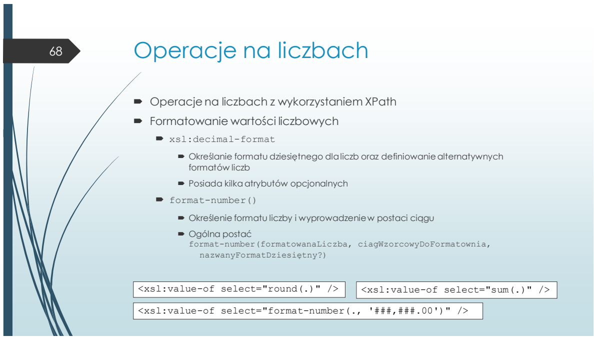
```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          version="1.0">
  <xsl:output method="html" encoding="windows-1250"/>
  <xsl:template match="/">
    <xsl:for-each select="/dokumentGlowny/pozycja">
            <xsl:apply-templates select="document(@plik)/SPIS" />
    </xsl:for-each>
  </xsl:template>
  <xsl:template match="SPIS">
    <H2>Opis książki</H2>
    <xsl:for-each select="KSIAZKA">
       <SPAN STYLE="font-style:italic">Tytuł: </SPAN>
       <xsl:value-of select="TYTUL"/><BR />
       <SPAN STYLE="font-style:italic">Autor: </SPAN>
       <xsl:value-of select="AUTOR"/><BR />
       <SPAN STYLE="font-style:italic">Rodzaj oprawy: </SPAN>
       <xsl:value-of select="OPRAWA"/><BR />
       <SPAN STYLE="font-style:italic">Liczba stron: </SPAN>
       <xsl:value-of select="STRONY"/><BR />
       <SPAN STYLE="font-style:italic">Cena: </SPAN>
       <xsl:value-of select="CENA"/><P />
    </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

Opis książki

Tytuł: Kubiś Puchatek
Autor: Allan Milne
Rodzaj oprawy: okładka papierowa
Liczba stron: 198
Cena: 52zł

Opis książki

Tytuł: Przygody Hucka
Autor: Mark Twain
Rodzaj oprawy: okładka papierowa
Liczba stron: 298
Cena: 72zł



68

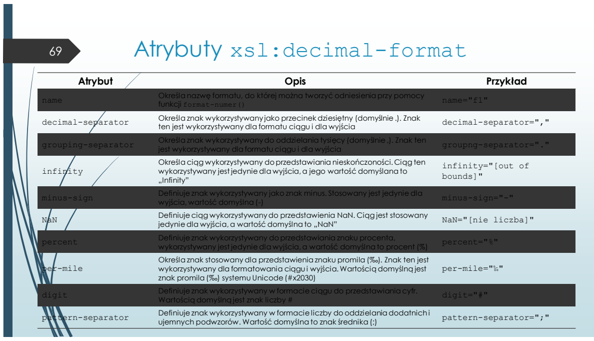
Operacje na liczbach

- Operacje na liczbach z wykorzystaniem XPath
- Formatowanie wartości liczbowych
  - `xsl:decimal-format`
    - Określanie formatu dziesiętnego dla liczb oraz definiowanie alternatywnych formatów liczb
    - Posiada kilka atrybutów opcjonalnych
  - `format-number()`
    - Określenie formatu liczby i wyprowadzenie w postaci ciągu
    - Ogólna postać
      `format-number(formatowanaLiczba, ciagWzorcowyDoFormatownia, nazwanyFormatDziesiętny?)`

```
<xsl:value-of select="round(.)" />
```

```
<xsl:value-of select="sum(.)" />
```

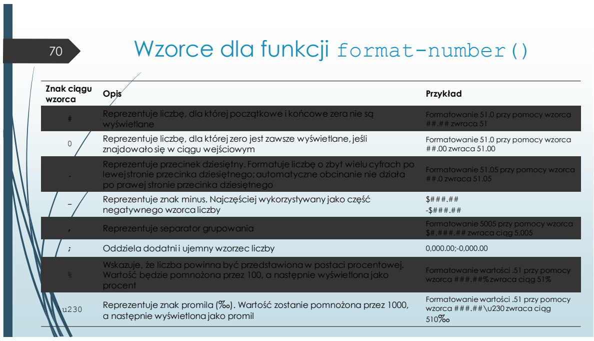
```
<xsl:value-of select="format-number(., '###,###.00')" />
```



69

Atrybuty `xsl:decimal-format`

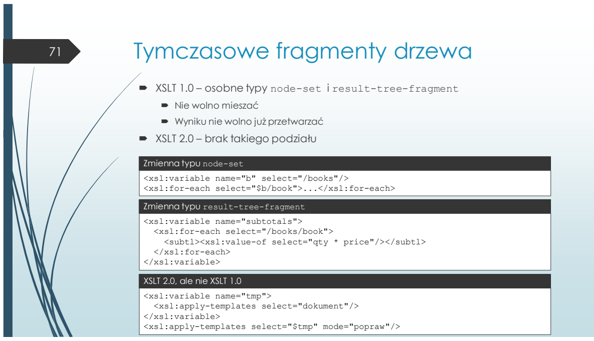
Atrybut	Opis	Przykład
`name`	Określa nazwę formatu, do której można tworzyć odniesienia przy pomocy funkcji `format-numer()`	`name="f1"`
`decimal-separator`	Określa znak wykorzystywany jako przecinek dziesiętny (domyślnie .). Znak ten jest wykorzystywany dla formatu ciągu i dla wyjścia	`decimal-separator=","`
`grouping-separator`	Określa znak wykorzystywany do oddzielania tysięcy (domyślnie ,). Znak ten jest wykorzystywany dla formatu ciągu i dla wyjścia	`groupng-separator="."`
`infinity`	Określa ciąg wykorzystywany do przedstawiania nieskończoności. Ciąg ten wykorzystywany jest jedynie dla wyjścia, a jego wartość domyślana to „Infinity"	`infinity="[out of bounds]"`
`minus-sign`	Definiuje znak wykorzystywany jako znak minus. Stosowany jest jedynie dla wyjścia, wartość domyślna (-)	`minus-sign="-"`
`NaN`	Definiuje ciąg wykorzystywany do przedstawienia NaN. Ciąg jest stosowany jedynie dla wyjścia, a wartość domyślna to „NaN"	`NaN="[nie liczba]"`
`percent`	Definiuje znak wykorzystywany do przedstawiania znaku procenta, wykorzystywany jest jedynie dla wyjścia, a wartość domyślna to procent (%)	`percent="%"`
`per-mile`	Określa znak stosowany dla przedstawienia znaku promila (‰). Znak ten jest wykorzystywany dla formatowania ciągu i wyjścia. Wartością domyślną jest znak promila (‰) systemu Unicode (#x2030)	`per-mile="‰"`
`digit`	Definiuje znak wykorzystywany w formacie ciągu do przedstawiania cyfr. Wartością domyślną jest znak liczby #	`digit="#"`
`pattern-separator`	Definiuje znak wykorzystywany w formacie liczby do oddzielania dodatnich i ujemnych podwzorów. Wartość domyślna to znak średnika (;)	`pattern-separator=";"`



70

Wzorce dla funkcji `format-number()`

Znak ciągu wzorca	Opis	Przykład
`#`	Reprezentuje liczbę, dla której początkowe i końcowe zera nie są wyświetlane	Formatowanie 51.0 przy pomocy wzorca ##.## zwraca 51
`0`	Reprezentuje liczbę, dla której zero jest zawsze wyświetlane, jeśli znajdowało się w ciągu wejściowym	Formatowanie 51.0 przy pomocy wzorca ##.00 zwraca 51.00
`.`	Reprezentuje przecinek dziesiętny. Formatuje liczbę o zbyt wielu cyfrach po lewej stronie przecinka dziesiętnego; automatyczne obcinanie nie działa po prawej stronie przecinka dziesiętnego	Formatowanie 51.05 przy pomocy wzorca ##.0 zwraca 51.05
`-`	Reprezentuje znak minus. Najczęściej wykorzystywany jako część negatywnego wzorca liczby	$###.## -$###.##
`,`	Reprezentuje separator grupowania	Formatowanie 5005 przy pomocy wzorca $#,###.## zwraca ciąg 5,005
`;`	Oddziela dodatni i ujemny wzorzec liczby	0,000.00;-0,000.00
`%`	Wskazuje, że liczba powinna być przedstawiona w postaci procentowej. Wartość będzie pomnożona przez 100, a następnie wyświetlona jako procent	Formatowanie wartości .51 przy pomocy wzorca ###.##% zwraca ciąg 51%
`\u230`	Reprezentuje znak promila (‰). Wartość zostanie pomnożona przez 1000, a następnie wyświetlona jako promil	Formatowanie wartości .51 przy pomocy wzorca ###.##\u230 zwraca ciąg 510‰



71

Tymczasowe fragmenty drzewa

- XSLT 1.0 – osobne typy `node-set` i `result-tree-fragment`
  - Nie wolno mieszać
  - Wyniku nie wolno już przetwarzać
- XSLT 2.0 – brak takiego podziału

Zmienna typu `node-set`

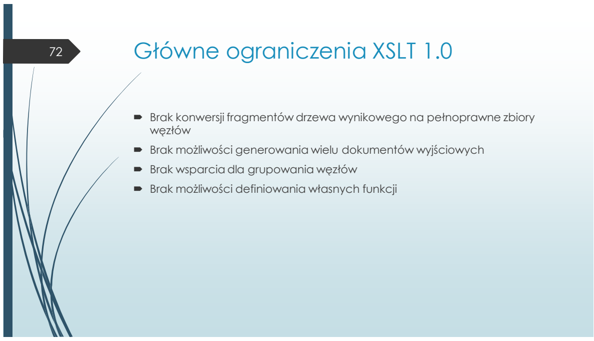
```
<xsl:variable name="b" select="/books"/>
<xsl:for-each select="$b/book">...</xsl:for-each>
```

Zmienna typu `result-tree-fragment`

```
<xsl:variable name="subtotals">
  <xsl:for-each select="/books/book">
    <subtl><xsl:value-of select="qty * price"/></subtl>
  </xsl:for-each>
</xsl:variable>
```

XSLT 2.0, ale nie XSLT 1.0

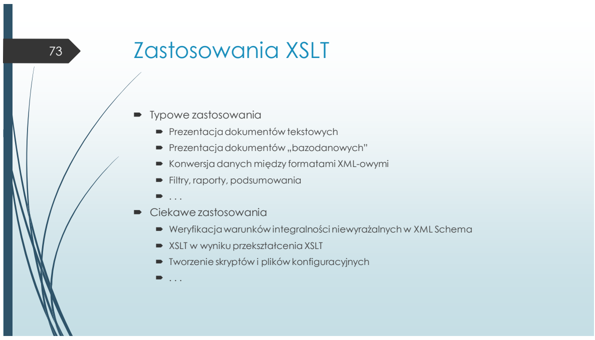
```
<xsl:variable name="tmp">
  <xsl:apply-templates select="dokument"/>
</xsl:variable>
<xsl:apply-templates select="$tmp" mode="popraw"/>
```



72

Główne ograniczenia XSLT 1.0

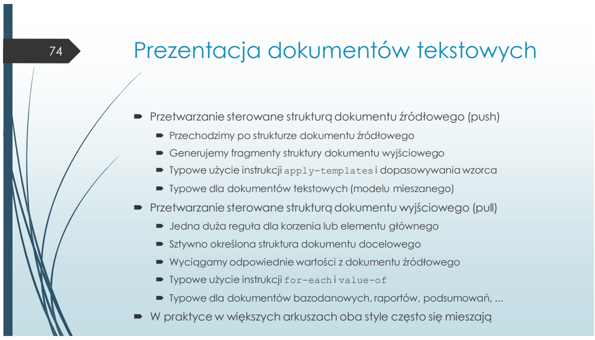
- Brak konwersji fragmentów drzewa wynikowego na pełnoprawne zbiory węzłów
- Brak możliwości generowania wielu dokumentów wyjściowych
- Brak wsparcia dla grupowania węzłów
- Brak możliwości definiowania własnych funkcji



73

Zastosowania XSLT

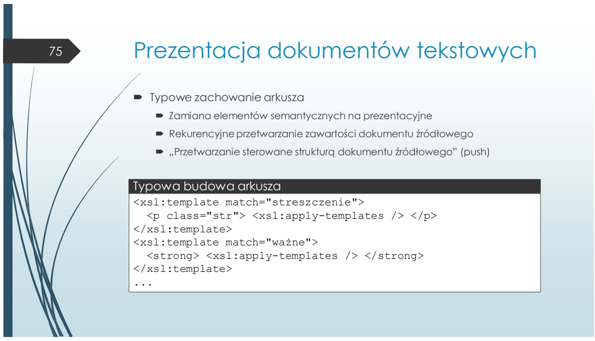
- Typowe zastosowania
  - Prezentacja dokumentów tekstowych
  - Prezentacja dokumentów „bazodanowych"
  - Konwersja danych między formatami XML-owymi
  - Filtry, raporty, podsumowania
  - . . .
- Ciekawe zastosowania
  - Weryfikacja warunków integralności niewyrażalnych w XML Schema
  - XSLT w wyniku przekształcenia XSLT
  - Tworzenie skryptów i plików konfiguracyjnych
  - . . .



74

Prezentacja dokumentów tekstowych

- Przetwarzanie sterowane strukturą dokumentu źródłowego (push)
  - Przechodzimy po strukturze dokumentu źródłowego
  - Generujemy fragmenty struktury dokumentu wyjściowego
  - Typowe użycie instrukcji `apply-templates` i dopasowywania wzorca
  - Typowe dla dokumentów tekstowych (modelu mieszanego)
- Przetwarzanie sterowane strukturą dokumentu wyjściowego (pull)
  - Jedna duża reguła dla korzenia lub elementu głównego
  - Sztywno określona struktura dokumentu docelowego
  - Wyciągamy odpowiednie wartości z dokumentu źródłowego
  - Typowe użycie instrukcji `for-each` i `value-of`
  - Typowe dla dokumentów bazodanowych, raportów, podsumowań, ...
- W praktyce w większych arkuszach oba style często się mieszają



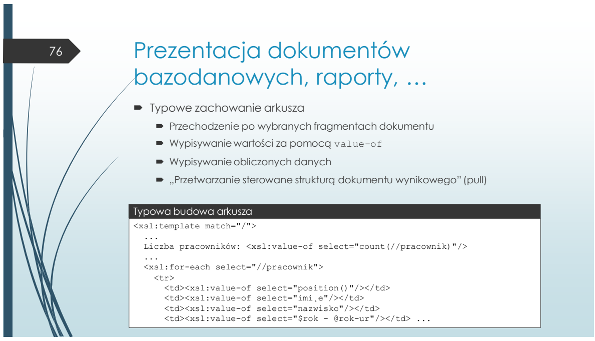
75

Prezentacja dokumentów tekstowych

- Typowe zachowanie arkusza
  - Zamiana elementów semantycznych na prezentacyjne
  - Rekurencyjne przetwarzanie zawartości dokumentu źródłowego
  - „Przetwarzanie sterowane strukturą dokumentu źródłowego" (push)

Typowa budowa arkusza

```
<xsl:template match="streszczenie">
  <p class="str"> <xsl:apply-templates /> </p>
</xsl:template>
<xsl:template match="ważne">
  <strong> <xsl:apply-templates /> </strong>
</xsl:template>
...
```



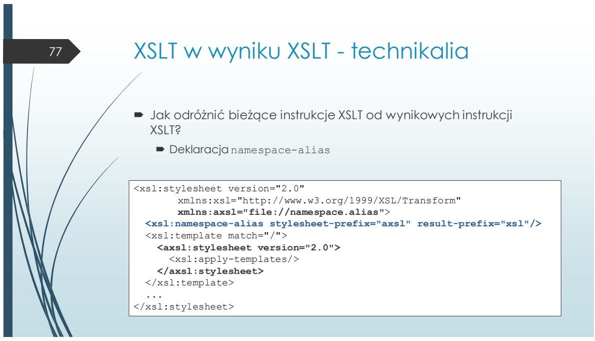
76

Prezentacja dokumentów bazodanowych, raporty, …

- Typowe zachowanie arkusza
  - Przechodzenie po wybranych fragmentach dokumentu
  - Wypisywanie wartości za pomocą `value-of`
  - Wypisywanie obliczonych danych
  - „Przetwarzanie sterowane strukturą dokumentu wynikowego" (pull)

Typowa budowa arkusza

```
<xsl:template match="/">
  ...
  Liczba pracowników: <xsl:value-of select="count(//pracownik)"/>
  ...
  <xsl:for-each select="//pracownik">
    <tr>
      <td><xsl:value-of select="position()"/></td>
      <td><xsl:value-of select="imi¸e"/></td>
      <td><xsl:value-of select="nazwisko"/></td>
      <td><xsl:value-of select="$rok - @rok-ur"/></td> ...
```



77

XSLT w wyniku XSLT - technikalia

- Jak odróżnić bieżące instrukcje XSLT od wynikowych instrukcji XSLT?
  - Deklaracja `namespace-alias`

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:axsl="file://namespace.alias">
  <xsl:namespace-alias stylesheet-prefix="axsl" result-prefix="xsl"/>
  <xsl:template match="/">
    <axsl:stylesheet version="2.0">
      <xsl:apply-templates/>
    </axsl:stylesheet>
  </xsl:template>
  ...
</xsl:stylesheet>
```

78

XSLT w wyniku XSLT – zastosowanie

- Schemat dokumentu
  - Opis struktury dokumentu
  - Etykiety i sposób formatowania
  - Przykład – opis formularza
- Pierwsze przekształcenie
  - Ze schematu dokumentu tworzy konkretne przekształcenie
- Drugie przekształcenie
  - Dopasowane do konkretnych instancji dokumentów
- Instancja dokumentu
  - Zgodny ze schematem
  - Zawiera konkretne dane
  - Przykład – wypełniony formularz

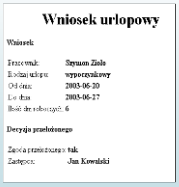
79

Przykład

Dane

```
<wniosek-urlopowy>
 <wniosek>
  <pracownik>Szymon Zioło</pracownik>
  <rodzaj>wypoczynkowy</rodzaj>
  <od>2003-06-20</od>
  <do>2003-06-27</do>
  <dni-roboczych>6</dni-roboczych>
 </wniosek>
 <decyzja>
  <zgoda>1</zgoda>
  <zastępca>Jan Kowalski</zastępca>
 </decyzja>
</wniosek-urlopowy>
```

Dokument



Wniosek urlopowy

Wniosek

Pracownik:	**Szymon Zioło**
Rodzaj urlopu:	**wypoczynkowy**
Od dnia:	**2003-06-20**
Do dnia:	**2003-06-27**
Ilość dni roboczych:	**6**

Decyzja przełożonego

Zgoda przełożonego:	**tak**
Zastępca:	**Jan Kowalski**

Źródło: Zioło, Sz., *XSLT do kwadratu*, Software 2.0, nr 6/2003

80

Przykład

- Zapisanie metainformacji w szablonie
- Generowanie przekształcenia poprzez szablon

```
<dokument nazwa="wniosek-urlopowy"
          etykieta="Wniosek urlopowy">
  <sekcja nazwa="wniosek" etykieta="Wniosek">
    <pole nazwa="pracownik" etykieta="Pracownik:"/>
    <pole nazwa="rodzaj" etykieta="Rodzaj urlopu:"/>
    <pole nazwa="od" etykieta="Od dnia:"/>
    <pole nazwa="do" etykieta="Do dnia:"/>
    <pole nazwa="dni-roboczych"
          etykieta="Ilość dni roboczych:"/>
  </sekcja>
  <sekcja nazwa="decyzja" etykieta="Decyzja przełożonego">
    <pole nazwa="zgoda" etykieta="Zgoda przełożonego:"
          typ="boolean"/>
    <pole nazwa="zastępca" etykieta="Zastępca:"/>
  </sekcja>
</dokument>
```

81

```
<xsl:stylesheet version="1.0"
          xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          xmlns:res="http://www.w3.org/1999/XSL/TransformAlias">

  <xsl:namespace-alias stylesheet-prefix="res" result-prefix="xsl"/>

  <xsl:template match="/">
    <res:stylesheet version="1.0">
      <res:output method="html"/>
      <xsl:apply-templates/>
    </res:stylesheet>
  </xsl:template>

  <xsl:template match="sekcja">
    <res:template match="{@nazwa}">
      <p><b><xsl:value-of select="@etykieta"/></b></p>
      <table><res:apply-templates/></table>
    </res:template>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="pole">
    <res:template match="{@nazwa}">
      <tr><td><xsl:value-of select="@etykieta"/></td>
      <td><b>
        <xsl:choose>
          <xsl:when test="@typ='boolean'">
            <res:choose>
              <res:when test="text()='1'">tak</res:when>
              <res:otherwise>nie</res:otherwise>
            </res:choose>
          </xsl:when>
          <xsl:otherwise>
            <res:value-of select="text()"/>
          </xsl:otherwise>
        </xsl:choose>
      </b></td></tr>
    </res:template>
    <xsl:apply-templates/>
  </xsl:template>
</xsl:stylesheet>
```