

# XSLT

Patryk Czarnik

Instytut Informatyki UW

XML i nowoczesne technologie zarządzania treścią – 2009/10

## 1. Możliwości XSLT

- Idea
- Instrukcje sterujące
- Sortowanie i grupowanie
- Tworzenie wyniku
- Zmienne i parametry
- Szablony nazwane i funkcje
- Metody serializacji
- Zastosowania

## 1. Możliwości XSLT

- Idea
- Instrukcje sterujące
- Sortowanie i grupowanie
- Tworzenie wyniku
- Zmienne i parametry
- Szablony nazwane i funkcje
- Metody serializacji
- Zastosowania

# XSLT – status

- Powstał w ramach standardu XSL.
- Zastosowania wykraczają poza prezentację 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.

# XSLT – status

- Powstał w ramach standardu XSL.
- Zastosowania wykraczają poza prezentację 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.

# XSLT – status

- Powstał w ramach standardu XSL.
- Zastosowania wykraczają poza prezentację 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.

# XSLT – status

- Powstał w ramach standardu XSL.
- Zastosowania wykraczają poza prezentację 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.

# 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,
  - ...

# 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,
  - ...

# XSLT – struktura arkusza

- **Arkusz** (*stylesheet*) składa się z szablonów.
- **Szablon** (*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 także:
  - wewnątrz wielu instrukcji XSLT,
  - ciało funkcji,
  - wartościowanie zmiennych i parametrów,

# XSLT – struktura arkusza

- **Arkusz** (*stylesheet*) składa się z szablonów.
- **Szablon** (*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 także:
  - wewnątrz wielu instrukcji XSLT,
  - ciało funkcji,
  - wartościowanie zmiennych i parametrów,

# XSLT – struktura arkusza

- **Arkusz** (*stylesheet*) składa się z szablonów.
- **Szablon** (*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 także:
  - wewnątrz wielu instrukcji XSLT,
  - ciało funkcji,
  - wartościowanie zmiennych i parametrów,

# XSLT – struktura arkusza

- **Arkusz** (*stylesheet*) składa się z szablonów.
- **Szablon** (*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 także:
  - wewnątrz wielu instrukcji XSLT,
  - ciało funkcji,
  - wartościowanie zmiennych i parametrów,

# Struktura arkusza – przykład

- element główny

```
<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="styeshheet" 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>
```

# Struktura arkusza – przykład

- deklaracje, „konfiguracja”

```
<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.xml"/>
  <xsl:param name="css"/>

  <xsl:template match="/">
    <html>
      <head><link rel="stylesheet" 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>
```

# Struktura arkusza – przykład

- szablony

```
<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="stylesheet" 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>
```

# Struktura arkusza – przykład

- 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.xml"/>
  <xsl:param name="css"/>

  <xsl:template match="/">
    <html>
      <head><link rel="stylesheet" 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>
```

# 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.

# 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.

# 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.

# 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.

# 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.

# Dopasowywanie szablonów

## Szablon – `template`

- przekształcenie jednego węzła we fragment wyniku
- atrybut `match` – do jakich węzłów pasuje szablon

## Wywołanie – `apply-templates`

- dla węzłów uruchomienie pasujących do nich szablonów
- opcjonalny atrybut `select` – dla jakich węzłów (domyślnie dzieci)

```
<xsl:template match="oddział">
  <ul>
    <xsl:apply-templates select="pracownik"/>
  </ul>
</xsl:template>
```

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

# Wybór szablonu

## Wzorce

- Zawarte w atrybutach `match` szablonów.
- Ograniczona postać ścieżek XPath.
- Osie tylko **wgłąb** (`child` i `attribute`).

## Dobór szablonu do węzła

- Węzeł musi „pasować” do wzorca.
- Spośród wielu pasujących wybierany ten o najściślej podanym `match` (formalny algorytm w rekomendacji).
- Możliwość ręcznego podania `priority`.
- Konflikt – błąd lub wybierany późniejszy szablon (zależne od implementacji),

# Wybór szablonu

## Wzorce

- Zawarte w atrybutach `match` szablonów.
- Ograniczona postać ścieżek XPath.
- Osie tylko **wgłąb** (`child` i `attribute`).

## Dobór szablonu do węzła

- Węzeł musi „pasować” do wzorca.
- Spośród wielu pasujących wybierany ten o najściślej podanym `match` (formalny algorytm w rekomendacji).
- Możliwość ręcznego podania `priority`.
- Konflikt – błąd lub wybierany późniejszy szablon (zależne od implementacji),

# Tryby przetwarzania (*modes*)

```
<xsl:template match="osoby">
  <table> ...
    <xsl:apply-templates select="osoba" mode="tabela"/>
  </table>
</xsl:template>

<xsl:template match="osoba">
  <li><xsl:apply-templates select="imię | nazwisko"/></li>
</xsl:template>

<xsl:template match="osoba" mode="tabela">
  <tr><td><xsl:value-of select="imię"/></td>
    <td><xsl:value-of select="nazwisko"/></td></tr>
</xsl:template>
```

## Możliwe wartości atrybutu mode

- nazwa
- #current – przy wywołaniu
- #default
- #all – w szablonie

# Tryby przetwarzania (*modes*)

```
<xsl:template match="osoby">
  <table> ...
    <xsl:apply-templates select="osoba" mode="tabela"/>
  </table>
</xsl:template>

<xsl:template match="osoba">
  <li><xsl:apply-templates select="imię | nazwisko"/></li>
</xsl:template>

<xsl:template match="osoba" mode="tabela">
  <tr><td><xsl:value-of select="imię"/></td>
    <td><xsl:value-of select="nazwisko"/></td></tr>
</xsl:template>
```

## Możliwe wartości atrybutu `mode`

- nazwa
- #default
- #current – przy wywołaniu
- #all – w szablonie

# 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.

# 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:
  - kopiuje wartość atrybutu do wyniku (wstawia węzeł tekstowy).
- Dla węzłów tekstowych:
  - kopiuje tekst do wyniku (wstawia węzeł tekstowy).
- Dla instrukcji przetwarzania i komentarzy:
  - nie rób nic.

# 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.

# 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.

# 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.

# Szablony wbudowane

## Dokument i element

```
<xsl:template match="element()|document()" mode="#all">
  <xsl:param .../> ...
  <xsl:apply-templates select="child::node()" mode="#current">
    <xsl:with-param .../> ...
  </xsl:apply-templates>
</xsl:template>
```

## Węzeł tekstowy

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

## Instrukcja przetwarzania i komentarz

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

# Szablony wbudowane

## Dokument i element

```
<xsl:template match="element()|document()" mode="#all">
  <xsl:param .../> ...
  <xsl:apply-templates select="child::node()" mode="#current">
    <xsl:with-param .../> ...
  </xsl:apply-templates>
</xsl:template>
```

## Węzeł tekstowy

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

## Instrukcja przetwarzania i komentarz

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

# Szablony wbudowane

## Dokument i element

```
<xsl:template match="element()|document()" mode="#all">
  <xsl:param .../> ...
  <xsl:apply-templates select="child::node()" mode="#current">
    <xsl:with-param .../> ...
  </xsl:apply-templates>
</xsl:template>
```

## Węzeł tekstowy

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

## Instrukcja przetwarzania i komentarz

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

# Pętla po sekwencji

## Instrukcja `for-each`

- Przechodzenie wszystkich węzłów wyliczonych przez `select`.
- W XSLT 2.0 przechodzenie dowolnej sekwencji (np. liczb).

## Przykład

```
<xsl:template match="pracownicy">
  <ul>
    <xsl:for-each select="pracownik">
      <li><xsl:value-of select="imię"/>
        <xsl:value-of select="nazwisko"/></li>
    </xsl:for-each>
  </ul>
</xsl:template>
```

# Zachowanie warunkowe – jedna gałąź

## Instrukcja `if`

- Sprawdzenie warunku logicznego z `test` (*Effective Boolean Value*).
- Jeśli prawdziwy, obliczenie i wstawienie wyniku.
- Brak `else`'a.

## Przykład

```
<xsl:template match="rozdział">
  <xsl:if test="@tytuł">
    <h2>Tytuł: <xsl:value-of select="@tytuł"/></h2>
  </xsl:if>
  <xsl:apply-templates />
</xsl:template>
```

# Zachowanie warunkowe – wiele gałęzi

## Instrukcja `choose`

- Dozory (`test`) wyliczane po kolei.
- Wybierana jedna gałąź – pierwsza z prawdziwym dozorem.
- Opcjonalna fraza `otherwise`.

## Przykład

```
<xsl:template match="konto">
Saldo konta jest
  <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:template>
```

# Sortowanie podczas przetwarzania

## Instrukcja `sort`

- Można użyć w `for-each`, `for-each-group` i `apply-templates`.
- Atrybuty – opcje sortowania:

```
select klucz,  
data-type rodzaj danych (text / number),  
order, case-order, stable, lang
```

## Przykład

```
<xsl:template match="wyniki-klasówki">  
  <ul>  
    <xsl:apply-templates select="student">  
      <xsl:sort select="punkty" data-type="number" order="descending"/>  
      <xsl:sort select="nazwisko" data-type="text"/>  
    </xsl:apply-templates>  
  </ul>  
</xsl:template>
```

# Sortowanie dowolnych sekwencji

Instrukcja `perform-sort`

## Przykład z rekomendacji

```
<xsl:function name="bib:books-by-price" as="schema-element (bib:book) *">
  <xsl:param name="in" as="schema-element (bib:book) *"/>
  <xsl:perform-sort select="$in">
    <xsl:sort select="xs:decimal (bib:price)"/>
  </xsl:perform-sort>
</xsl:function>

...

<xsl:copy-of select="bib:books-by-price (//bib:book)
  [position() = 1 to 5]"/>
```

# 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

# 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

# Grupowanie – przykłady

## 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>
```

# Grupowanie – przykłady

## 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>
```

## Węzły drzewa wynikowego

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

# 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.
- Węzły tekstowe z „czarnymi” znakami.

```
<xsl:template match="pracownik">
  <div>
    <xsl:if test="parent::dział/nazwa = 'księgowość'">
      
    </xsl:if>
    Pracownik
  <xsl:apply-templates />
  <!-- Tego nie będzie w wyniku -->
</div>
</xsl:template>
```

# 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>
```

# Instrukcje tworzące węzły – typowe zastosowania

- **Wstawienie instrukcji przetwarzania lub komentarza.**
- Wstawienie samych białych znaków.
- Wstawienie tekstu bez nadmiarowych białych znaków.
- Dynamicznie określona nazwa elementu lub atrybutu.
- Warunkowe wstawienie atrybutu.

## Przykłady

```
<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>
```

# Instrukcje tworzące węzły – typowe zastosowania

- Wstawienie instrukcji przetwarzania lub komentarza.
- **Wstawienie samych białych znaków.**
- Wstawienie tekstu bez nadmiarowych białych znaków.
- Dynamicznie określona nazwa elementu lub atrybutu.
- Warunkowe wstawienie atrybutu.

## Przykłady

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

# Instrukcje tworzące węzły – typowe zastosowania

- Wstawienie instrukcji przetwarzania lub komentarza.
- Wstawienie samych białych znaków.
- **Wstawienie tekstu bez nadmiarowych białych znaków.**
- Dynamicznie określona nazwa elementu lub atrybutu.
- Warunkowe wstawienie atrybutu.

## Przykłady

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

# Instrukcje tworzące węzły – typowe zastosowania

- Wstawienie instrukcji przetwarzania lub komentarza.
- Wstawienie samych białych znaków.
- Wstawienie tekstu bez nadmiarowych białych znaków.
- **Dynamicznie określona nazwa elementu lub atrybutu.**
- Warunkowe wstawienie atrybutu.

## Przykłady

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

# Instrukcje tworzące węzły – typowe zastosowania

- Wstawienie instrukcji przetwarzania lub komentarza.
- Wstawienie samych białych znaków.
- Wstawienie tekstu bez nadmiarowych białych znaków.
- Dynamicznie określona nazwa elementu lub atrybutu.
- **Warunkowe wstawienie atrybutu.**

## Przykłady

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

# Nazwane zbiory atrybutów

## Przykład definicji

```
<xsl:attribute-set name="ważne">  
  <xsl:attribute name="font-weight">bold</xsl:attribute>  
  <xsl:attribute name="color">red</xsl:attribute>  
</xsl:attribute-set>
```

## Przykład użycia

```
<xsl:template match="wyróżnienie">  
  <fo:inline xsl:use-attribute-sets="ważne">  
    <xsl:apply-templates />  
  </fo:inline>  
</xsl:template>  
  
<xsl:template match="uwaga">  
  <fo:block xsl:use-attribute-sets="ważne">  
    <xsl:apply-templates />  
  </fo:block>  
</xsl:template>
```

# Nazwane zbiory atrybutów

## Przykład definicji

```
<xsl:attribute-set name="ważne">  
  <xsl:attribute name="font-weight">bold</xsl:attribute>  
  <xsl:attribute name="color">red</xsl:attribute>  
</xsl:attribute-set>
```

## Przykład użycia

```
<xsl:template match="wyróżnienie">  
  <fo:inline xsl:use-attribute-sets="ważne">  
    <xsl:apply-templates />  
  </fo:inline>  
</xsl:template>  
  
<xsl:template match="uwaga">  
  <fo:block xsl:use-attribute-sets="ważne">  
    <xsl:apply-templates />  
  </fo:block>  
</xsl:template>
```

# Wstawianie wyniku wyrażenia XPath

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

# 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>
```

## 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>Patryk</imię><nazwisko>Czarnik</nazwisko></osoba>  
<osoba><imię>Maciej</imię><nazwisko>Ogrodniczuk</nazwisko></osoba>
```

### Arkusze

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

### Wynik

```
<wynik>Patryk</wynik>
```

## 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>Patryk</imię><nazwisko>Czarnik</nazwisko></osoba>  
<osoba><imię>Maciej</imię><nazwisko>Ogrodniczuk</nazwisko></osoba>
```

### Arkusz

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

### Wynik

```
<wynik>Patryk Maciej</wynik>
```

# Szablony wartości atrybutu

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.

```


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

# Szablony wartości atrybutu

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.

```

```

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

# Szablony wartości atrybutu

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.

```


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

# Zmienne lokalne

- Zmienne i parametry w XSLT → zmienne w XPath.
- Zmienne „deklaratywne” – brak instrukcji przypisania.
- Cykliczne referencje zabronione.

## Przykład

```
<xsl:template match="konto">
  <xsl:variable name="jake" >
    <xsl:choose>
      <xsl:when test="saldo > 0">dodatnie</xsl:when>
      <xsl:when test="saldo < 0">ujemne</xsl:when>
      <xsl:otherwise>równe zero</xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

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

# Konsekwencje deklaratywności zmiennych

## Zmienna niezdefiniowana w miejscu odwołania

```
<xsl:choose>
  <xsl:when test="saldo >= 0">
    <xsl:variable name="jakiel">nieujemne</xsl:variable>
  </xsl:when>
  <xsl:otherwise>
    <xsl:variable name="jakiel">ujemne</xsl:variable>
  </xsl:otherwise>
</xsl:choose>
```

Saldo konta jest <xsl:value-of select="\$jakiel"/>.

## Nowa zmienna tylko na chwilę zakrywa starą

```
<xsl:variable name="jakiel">nieujemne</xsl:variable>
<xsl:if test="saldo < 0">
  <xsl:variable name="jakiel">ujemne</xsl:variable>
</xsl:if>
```

Saldo konta jest <xsl:value-of select="\$jakiel"/>.

# 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 < 0">
  <xsl:variable name="jakie">ujemne</xsl:variable>
</xsl:if>
```

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

# Zmienne i parametry – globalne

- Widoczne w całym arkuszu.
- Zmienne globalne wyliczane raz na początku przekształcenia..
- Wartości parametrów przekazywane „z zewnątrz” (można podać wartość domyślną).

## Przykład

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

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

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

# Parametry szablonów

- W szablonie param.
- W wywołaniu with-param.

## Przykład

```
<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>
```

# Szablony nazwane

- W szablonie atrybut `name`.
- `call-template` uruchamia.
- Bez zmiany węzła bieżącego (inaczej niż `apply-templates`).
- Możliwa rekursja.

## Przykład

```
<xsl:template name="opisz-element">
  <p>Element o nazwie <xsl:value-of select="name()" />.</p>
</xsl:template>

<xsl:template match="/">
  <html><body>
    <h1>Wszystkie elementy:</h1>
    <xsl:for-each select="//*">
      <xsl:call-template name="opisz-element" />
    </xsl:for-each>
  </body></html>
</xsl:template>
```

# Parametry i rekursja w szablonach nazwanych

„Programowanie” w XSLT (nawet 1.0)

## Silnia (z akumulatorem)

```
<xsl:template name="silnia">
  <xsl:param name="n"/>
  <xsl:param name="res" select="1"/>
  <xsl:choose>
    <xsl:when test="$n > 1">
      <xsl:call-template name="silnia">
        <xsl:with-param name="n" select="$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>
```

# Definiowanie własnych funkcji (XSLT 2.0)

## Silnia (bez akumulatora)

```
<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>
```

# Serializacja wyniku

- Wynikiem przekształcenia drzewo XPath.
- Serializacja – zapisanie wyniku jako sekwencji bajtów.
- Metody serializacji:
  - xml,
  - html,
  - xhtml (tylko XSLT 2.0),
  - text.

## Specyfikacja w arkuszu

```
<xsl:output method="html" encoding="iso-8859-2"/>
```

# Dodatkowe parametry serializacji

- `encoding` – kodowanie znaków,
- `version` – wersja XML lub HTML,
- `doctype-public`, `doctype-system` – deklaracja DOCTYPE,
- `indent` – automatyczne wcięcia.

## Specyfikacja w arkuszu

```
<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"/>
```

## Początek wynikowego dokumentu

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

## Dodatkowe parametry serializacji

- `encoding` – kodowanie znaków,
- `version` – wersja XML lub HTML,
- `doctype-public`, `doctype-system` – deklaracja DOCTYPE,
- `indent` – automatyczne wcięcia.

### Specyfikacja w arkuszu

```
<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"/>
```

### Początek wynikowego dokumentu

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

# Zapisywanie dodatkowych plików

## Instrukcja `result-document`

Zapis do pliku dodatkowego drzewa wynikowego.

### Przykład z rekomendacji

```
<xsl:output name="section-format" method="xhtml" indent="no"/>

<xsl:for-each-group select="*/xhtml:body/*"
                  group-starting-with="xhtml:h1">
  <xsl:result-document href="section{position()}.html"
                    format="section-format" validation="strip">
    <html xmlns="http://www.w3.org/1999/xhtml">
      <head><title><xsl:value-of select="."/></title></head>
      <body>
        <xsl:copy-of select="current-group()" />
      </body> </html>
    </xsl:result-document>
  </xsl:for-each-group>
```

# Czego nie ma w XSLT 1.0

- XPath 2.0 z sekwencjami, `if`-em, typami XML Schema...
- Definiowania funkcji.
- Grupowania.
- Zapisywania dodatkowych plików.
- `analyze-string`.
- ...

# 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.

## 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"/>
```

# Co się nie zmieściło

- Numeracja.
- Komunikaty diagnostyczne i przechwytywanie błędów.
- Analiza tekstu wyrażeniem regularnym i czytanie plików tekstowych (XSLT 2.0).
- Obsługa schematów (XSLT 2.0).
- Serializacja a. . . :
  - przestrzenie nazw,
  - sekcje CDATA,
  - zamiana znaków.

# 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.
- ...

# 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.

# 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>
```

...

# 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="imię"/></td>
      <td><xsl:value-of select="nazwisko"/></td>
      <td><xsl:value-of select="$rok - @rok-ur"/></td>
    </tr>
  </xsl:for-each>
</xsl:template>
```

# XSLT w wyniku XSLT – technikalia

## Jak odróżnić bieżące instrukcje XSLT od wynikowych instrukcji XSLT?

- Deklaracja `namespace-alias`

### Przykładzik

```
<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>
```

# XSLT w wyniku XSLT – zastosowanie

## Schemat dokumentu

- Opis struktury dokumentu.
- Etykiety i sposób formatowania.
- Przykład – opis formularza.

## Instancja dokumentu

- Zgodny ze schematem.
- Zawiera konkretne dane.
- Przykład – wypełniony formularz

## Pierwsze przekształcenie

- Ze schematu dokumentu tworzy konkretne przekształcenie.

## Drugie przekształcenie

- Dopasowane do konkretnych instancji dokumentów.