

# XSLT

Patryk Czarnik

Instytut Informatyki UW

XML i nowoczesne technologie zarządzania treścią – 2008/09





# XSLT – status

- Wersja 1.0
  - listopad 1999
  - powiązane z XPath 1.0
- Wersja 2.0
  - styczeń 2007
  - powiązane z XPath 2.0
- Powstał w ramach *Extensible Stylesheet Language*.
- Zastosowania wykraczają poza prezentację XML.

# Budowa arkusza

- **Arkusz** zbudowany z **szablonów**.
- Elementy (z prz. nazw) XSLT – deklaracje i **instrukcje**.
- **Konstruktory sekwencji** – fragmenty traktowane jako wyrażenia
  - ciała szablonów, funkcji, wartościowania zmiennych i parametrów,
  - tworzą drzewo wynikowe,
  - mogą zawierać instrukcje XSLT.

# Struktura arkusza

- 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

- 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

- 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="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

- 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="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>
```

# Model przetwarzania

- Szablon – przekształcenie węzła we fragment drzewa
  - atrybut `match`: do jakich węzłów "pasuje" szablon.
- Początek przetwarzania – szablon pasujący do korzenia.
- `apply-templates` – rekurencyjne wywołania szablonów.
  - atrybut `select`: dla jakich węzłów szukać i stosować szablony,
  - brak `select` – `child::node().dzieci`.

## Przykład

```
<xsl:template match="osoby">
  <ul>
    <xsl:apply-templates select="osoba"/>
  </ul>
</xsl:template>
```

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

# Dopasowywanie wzorców

- Zawartość `match` – **wzorzec**:
  - ograniczona postać ścieżek XPath,
  - osie tylko włąb (`child` i `attribute`).
- Dla każdego węzła z wyniku `select`:
  - dobierany szablon o najściślej podanym `match`,
  - chyba że ręcznie podano `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,
- #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

```
<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>

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

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

## 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

- Elementy spoza przestrzeni nazw XSLT oraz węzły tekstowe znajdujące się w konstruktorach sekwencji są przepisywane do wyniku.
- Atrybuty tych elementów także.
- Zawartość elementów jest przetwarzana jako konstruktor sekwencji.

```
<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

- Węzły dowolnych rodzajów.
- Nazwy ustalane dynamicznie.
- Istnienie atrybutów ustalane dynamicznie.

```
<xsl:template match="pracownik">
  <xsl:element name="{if parent::dział/nazwa = 'księgowość'
    then 'księgowy' else 'pracownik'}">

    <xsl:if test="@stanowisko = 'kierownik'">
      <xsl:attribute name="szef">
        tak
      </xsl:attribute>
    </xsl:if>
    <xsl:text>Zawartość tekstowa
    <xsl:comment>To będzie komentarz
  </xsl:element>

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

# Umieszczanie 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!.

# Umieszczanie wyniku wyrażenia XPath – przykłady

```
<xsl:sequence select="for $i in (1 to 10) return $i * $i"/>
<xsl:sequence select="//pracownik[@stanowisko='handlowiec']"/>
<xsl:copy-of select="//pracownik[@stanowisko='handlowiec']"/>
<xsl:value-of select="//pracownik[@stanowisko='handlowiec']/imię"/>
<xsl:value-of>
  <xsl:choose>
    <xsl:when test="$x > 0">Większe</xsl:when>
    <xsl:when test="$x < 0">Mniejsze</xsl:when>
  </xsl:choose>
</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**.
- To źródło częstych błędów :)

### Dokument

```
<osoba><imię>Patryk</imię><nazwisko>Czarnik</nazwisko></osoba>  
<osoba><imię>Szymon</imię><nazwisko>Zioło</nazwisko></osoba>
```

### Arkusze

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

### Wynik

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

## value-of w XSLT 2.0

- Sekwencja poddana atomizacji.
- Wynikowy tekst:
  - rzutowanie każdego atomu na `string`
  - rozdzielone separatorem podanym w atrybucie `separator`
  - jeśli brak tego atrybutu — rozdzielone spacjami.

### Dokument

```
<osoba><imię>Patryk</imię><nazwisko>Czarnik</nazwisko></osoba>  
<osoba><imię>Szymon</imię><nazwisko>Zioło</nazwisko></osoba>
```

### Arkusze

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

### Wynik

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

# Szablony 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
  - wrażenie 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

- 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>
```

# Instrukcja `for-each`

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

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

# Instrukcja `if`

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

```
<xsl:template match="rozdział">
  <xsl:if test="tytuł">
    <h2>Rozdział <xsl:value-of select="tytuł"/></h2>
  </xsl:if>
  <xsl:apply-templates select="node() except tytuł"/>
</xsl:template>
```

## Instrukcja `choose`

- Podobnie jak `if`, ale wiele dozorów.
- Dozory (`test` w `when`) wyliczane w kolejności gałęzi `when`.
- Wybierana jedna gałąź – pierwsza z prawdziwym dozorem.
- Opcjonalna fraza `otherwise`.

```
<xsl:template match="liczby">
  <xsl:choose>
    <xsl:when test="$x > 0"> Większe </xsl:when>
    <xsl:when test="$x < 0"> Mniejsze </xsl:when>
    <xsl:otherwise> Chyba równe... </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

# Zmienne lokalne

- Zmienne „deklaratywne” – brak instrukcji przypisania.
- Cykliczne referencje zabronione.

## Przykład

```
<xsl:template match="cośtam">
  <xsl:variable name="jaki_x">
    <xsl:choose>
      <xsl:when test="$x > 0">dodatni</xsl:when>
      <xsl:when test="$x = 0">równy zero</xsl:when>
      <xsl:when test="$x < 0">ujemny</xsl:when>
    </xsl:choose>
  </xsl:variable>

  ...x jest <xsl:value-of select="$jaki_x"/>...
</xsl:template>
```

# Konsekwencje deklaratywności zmiennych

## Zmienna niezdefiniowana w miejscu odwołania

```
<xsl:choose>
  <xsl:when test="$x > 0">
    <xsl:variable name="jaki_x">dotatni</xsl:variable>
  </xsl:when>
  <xsl:otherwise>
    <xsl:variable name="jaki_x">nie dodatni</xsl:variable>
  </xsl:otherwise>
</xsl:choose>
```

...x jest <xsl:value-of select="\$jaki\_x"/>...

## Nowa zmienna tylko na chwilę zakrywa starą

```
<xsl:variable name="jaki_x">nie dodatni</xsl:variable>
<xsl:if test="$x > 0">
  <xsl:variable name="jaki_x">dotatni</xsl:variable>
</xsl:if>
```

...x jest <xsl:value-of select="\$jaki\_x"/>...

# Konsekwencje deklaratywności zmiennych

## Zmienna niezdefiniowana w miejscu odwołania

```
<xsl:choose>
  <xsl:when test="$x > 0">
    <xsl:variable name="jaki_x"> dodatni</xsl:variable>
  </xsl:when>
  <xsl:otherwise>
    <xsl:variable name="jaki_x"> nie dodatni</xsl:variable>
  </xsl:otherwise>
</xsl:choose>
```

```
...x jest <xsl:value-of select="$jaki_x"/>...
```

## Nowa zmienna tylko na chwilę zakrywa starą

```
<xsl:variable name="jaki_x"> nie dodatni</xsl:variable>
<xsl:if test="$x > 0">
  <xsl:variable name="jaki_x"> dodatni</xsl:variable>
</xsl:if>
```

```
...x jest <xsl:value-of select="$jaki_x"/>...
```

## Zmienne i parametry – globalne

- Wartość **parametru** przekazana "z zewnątrz" podczas wywołania (można podać wartość domyślną).
- Zmienne wyliczane raz dla całego arkusza.

### 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

- Pozwalają na "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

- Tylko w 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>
```

# Sortowanie podczas przetwarzania

- Instrukcja `sort` w `for-each`, `for-each-group` i `apply-templates`.
- Opcje sortowania w atrybutach:
  - `select` klucz,
  - `data-type` rodzaj danych,
  - `order`, `case-order`, `stable`, `lang`

## Prosty 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 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 (tylko 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 aktualnej grupy (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>Nazwiska:
    <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>Nazwiska:
    <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>
```



## Analiza tekstu (tylko 2.0)

- Instrukcja `analyze-string`
  - atrybut `select` – źródło tekstu (rzutowane na `string`),
  - atrybut `regex` – wyrażenie do którego dopasowujemy tekst.
- Funkcja `regex-group`.

### Przykład z rekomendacji

```
<xsl:analyze-string select="body" regex="\[(.*)\]">
  <xsl:matching-substring>
    <cite><xsl:value-of select="regex-group(1)"/></cite>
  </xsl:matching-substring>
  <xsl:non-matching-substring>
    <xsl:value-of select="."/>
  </xsl:non-matching-substring>
</xsl:analyze-string>
```

# Czytanie dokumentów tekstowych (tylko 2.0)

- Funkcja `unparsed-text`

## Przykład nieprzetestowany

```
<xsl:analyze-string regex="([\^,]*) , (|$) "  
  select="unparsed-text(' dane.csv', 'windows-1250') ">  
  <xsl:matching-substring>  
    <td><xsl:value-of select="regex-group(1) " /></td>  
  </xsl:matching-substring>  
</xsl:analyze-string>
```

# 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.

## 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"/>
```

## Efekt

```
<?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.

### 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"/>
```

### Efekt

```
<?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`.
- Tworzy i zapisuje do pliku dodatkowe drzewo wynikowe.

## 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, typami, `if`-em itd.
- Definiowania funkcji.
- Grupowania.
- Zapisywania dodatkowych plików z wynikiem.
- `analyze-string`.
- ...

# Tymczasowe fragmenty drzewa

- XSLT 1.0 – osobne typy node-set i result-tree-fragment.
  - nie wolno mieszać,
  - r-t-f nie wolno przetwarzać.
- XSLT 2.0 brak takiego podziału.

## XSLT 2.0, ale nie XSLT 1.0

```
<xsl:variable name="tmp">  
  <xsl:apply-templates select="cośtam"/>  
</xsl:variable>
```

```
<xsl:apply-templates select="$tmp" mode="blabla"/>
```

# Dwa style przetwarzania

- 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.