10. XSLT Stylesheets

These exercises are about XSLT stylesheets. You are asked to write a stylesheets to achieve a number of tasks.
Run your stylesheets using Kernow. On the sheet for Lab 9, you can find instructions how to download Kernow, a graphical frontend to the Saxon XQuery/XSLT processor. Kernow is a Java package and can be run under Linux, Windows and Mac OS.
You can develop your stylesheets in the XSLT Sandbox of Kernow. The outermost element of your stylesheets should be of the kind `xsl:stylesheet`, with attributes set as shown below:

```
<xsl:stylesheet version="2.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

In the lectures, we have introduced XSLT 1.0, because version 2.0 is not yet widely supported. Saxon, however, is a processor for XSLT 2.0 and will return a warning if you declare your stylesheet as of version 1.0.
To tell Kernow that the output should be XML 1.0 and that we would like it properly indented, add as the first element in the stylesheet the following output specification:

```
<xsl:output method="xml"
            version="1.0"
            encoding="iso-8859-1"
            indent="yes"/>
```

The version of Kernow that you download allows you to run 100 queries. Then it will ask you to enter a code in order to proceed. I have the code and will send it to you if you ask me by email.
These exercises are largely similar to the ones for XQuery. This is intentional, since both XQuery and XSLT allow one to specify transformations turning XML documents into other XML or (X)HTML documents.
The exercises allow you to compare the means by which such transformations can be expressed in the two languages.

1. Recipes

The following transformations are over the `recipes.xml` document. Formulate your transformations in such a way that they return the correct result for all possible recipe documents that satisfy `recipes.dtd`.

1. Return a list, inside an element <`recipes`>, of recipes, containing for every recipe the recipe's title element and an element with the number of calories. Use different approaches:

   (a) express iteration by recursive calls of templates

   (b) express iteration by <`xsl:for-each`> elements.

   Create new elements

   (a) by explicit construction, that is by writing the tags into the code,

   (b) by dynamic construction, that is, by using <`xsl:element`> and <`xsl:attribute`> elements,

   (c) by shallow and deep copying, wherever the latter is possible.

2. Using iteration by recursion, return a similar list, alphabetically ordered according to title.

3. Using iteration by means of <`xsl:for-each`>, return a similar list, ordered according to calories in descending order.

   (**Hint:** Consult the Web, e.g. the site `www.w3schools.com`, if you encounter difficulties with the ordering.)

4. Return a similar list, with title as attribute and calories as content.

5. Return a list, inside an element <`recipes`>, of recipes, where each recipe contains the title and the top level ingredients, while dropping the lower level ingredients.

2. Countries

The following are transformations over the `countries.xml` document. Formulate your transformations in such a way that they return the correct result for all possible recipe documents that satisfy `countries.dtd`.

1. Return an element <`countries`> with a list of <`country`> elements, containing in turn elements <`name`> and <`population`>, such that each country of the document appears once. Write the stylesheet so that iteration is achieved by recursive calls to templates.

2. Return a list of <city> elements, containing the name of the city, such that each city has an attribute `population` and another attribute `country`. The cities are returned according to their population, in descending order. Use iteration by template call.

3. Restructure the document by listing countries according to population, cities within each country according to population, and languages within each country according to percentage, all in descending order.

4. Return an element <languages> with a list of <language> elements, alphabetically sorted, where each language element containts a list of country elements, such that the language is spoken in the country, together with the number of speakers of the language in that country.

   Hint: You may need parameters, named templates, calls to templates, and the formating function `format-number`.