Task documentation XQR: XML Query in Python3.6 for IPP 2016/2017

Name and Surname: Ján Gula

Login: xgulaj00

Task

Our task was to create a simple query language similar to the well-known SQL but using only the SELECT command. The query then filtered data written in an XML file which was our input. The result was then written into an output XML file. This program acts as a script. My implementation is mostly build on regular expressions making it much shorter than other versions.

Parsing input arguments

To begin with our script had to analyze input arguments given from the user. Usage of input parameters is described in the `--help` parameter. My implementation uses the `argparse` library to store the values of arguments and to check if each parameter is valid or if the parameter is set only once. If the user sets the `--help` parameter we need to check if it's the only parameter given. If not an error is returned.

Implementing input arguments

Argument parser from the first part set specific variables to the values given from the command line. Each argument was compared to the built-in variable `None` which determined if the argument was or was not set. The hardest argument to implement was the --query argument which is explained below. Some arguments for example --n and --root were only few write commands.

Query parsing

Secondly, our file had to analyze the input query which could be inserted in two ways, via the --query argument or via the --qf argument which determined a file with a query of the same format. My implementation separates the whole query into different parts using a regular expression. First part is the text after SELECT which is obligatory, second part is the same but after the keyword FROM and then there are optional parts WHERE and LIMIT which were captured by groups in regular expressions if present.

Each part was then analyzed separately. The most complicated was the WHERE part where we had to test a given condition. I've solved this problem with one function `conditionResult` returning a Boolean value. The input of this function was the text following our WHERE clause separated into strings. In this function I've also checked for query errors for example CONTAINS could only be followed by a string.

Another problem was to recognize if our FROM and WHERE clause is follow by an element or an attribute or a combination of these two. My implementation solves this problem by using an else if tree defining solutions for each option. The WHERE clause could also start with an infinite number of NOT, negating the condition followed by our WHERE clause. My implementation solves this by a regular expression searching for the word NOT and afterwards separating the string into an array. After that we could simply use the built-in count function to find out how many NOT's were inserted. If the number was dividable by two we could assume that the condition will maintain its value. If not we knew that we should negate the condition result.

Formatting output

My implementation uses the `xml.dom.minidom.parse` function with the `getElementsByTagName` function. To check if an attribute is present I'm using `hasAttribute` and to get the value of an attribute there is a `getAttribute` function returning the value of our attribute. If we had the value of an element or an attribute and wanted an XML representation we could simply use the `toxml` function. If an argument requiring the formatting of our output was present I've used the simple `write` command to print text into our output file these were for example `--n` which required us not to print the header of our XML file and the argument `--`

`root` which stated that the result of our query should be enfolded in an XML element with the name of the argument given.