Task documentation XQR: XML query in Python 3.6 for IPP 2016/2017
Name and surname: Adrián Tóth
Login: xtotha01

My solution of this project can be divided into these following parts: analysis of parameters, input/output managing, loading the query, lexical analysis of query, syntactic analysis of query, executing the query. Structure of the script is based mainly on function calls and a little bit on classes. The source code of the script can be divided into parts too. At the beginning, there are import statements. After them, there are function definitions and class definitions. At the end, there is a part, where functions or constructors of classes are called.

1.) Analysis of parameters:

When the script is launched, there is a class call, which will provide analysis of parameters. Class, which is representing pseudo argparse implemented by myself, provide check of parameters and set up class variables for the next use. It does duplicity check and supported parameter check. On finding a wrong parameter, it will proceed an error function call, which writes the correct message to stderr and terminates the script with appertaining exitcode. Some of the exceptions are checked after the parameter analysis. Exception, for example, mean: parameter *--help*, parameters *--qf* and *--query* at the same time, missing query parameter (one parameter from *--qf* or *--query* is missing).

2.) Input/output managing:

After the analysis of parameters, the script will set up input and output.

Firstly, the input is set up. If there was an *--input* parameter, we check the validity of the file name. If the file name was correct and the file is readable, we can set the stdin from file via codecs, precisely via codecs.open() function. If there was not given an *--input* parameter, we do nothing, because the data are from stdin.

Secondly, we have to set up the output. If there was an *--output* parameter, then we check the validity of file name. If the file name is valid and writeable, then the stdout is redirected to this file. Otherwise, if there was *--output* parameter with filename, but the file does not exist, we create this file and redirect the stdout to this file. If there was not *--output* parameter, we redirect every data to stdout.

3.) Loading the query:

Query has to be loaded from a file or as a parameter value. File is specified via parameter *--qf* which includes the query. Query, as a parameter value, is specified in *--query* parameter. Depending on parameters, we load the query into one variable as a string. Every following correction check of the query, is executed on this string variable.

4.) Lexical analysis of query:

The query is passed to the scanner which will provide lexical analysis of the loaded query. Scanner works in steps. The input string, including query, is iterated through character by character in a loop. Inside of this loop, we append characters into a string, representing a buffer, until a whitespace is found. Then we pass the buffer content to a function which checks the lexeme and creates a token for it.

We have to make a special token “end token” because of syntactic analysis. This problem is solved with appending a special character “}” which is evaluated as the end token. The loop ends, when the last character is “~” what is the end of string character, which is appended to the query.

5.) Syntactic analysis of query:

Syntactic analyser is based on LL grammar from the task, implemented as recursive-descent parsing. The requirements for the parser is scanner. Scanner is used for calling for the next token via method. Parser is called via first function representing the first rule of LL grammar. Every other function representing other LL rules is called inside of this function. If these functions do not find any syntactic errors, they return true value. If the first function returns true value we can judge the query as syntactically correct. After syntactic analysis of the query, the query is passed to the next part, where we execute the query.

6.) Executing the query:

The query is executed in parts, via xml.dom.minidom module´s functions.

Firstly, we execute the “FROM” clause trough the xml.dom.minidom´s function and we save the result to an array. Afterwards we execute the “SELECT” clause. We iterate trough element by element in the array, saving those ones which meet the criteria of “SELECT” clause.

If there was “WHERE” clause, we iterate trough the array eliminating those elements, which do not meet the criteria of the appertaining condition. The “WHERE” clause is using an additional function which provide a condition evaluation depending of operator.

The last part of executing the query is the “LIMIT” clause. We retain the count of elements specified in “LIMIT” clause.

After the execution of the query we have to print out the saved data. We have correct data in array which meet the criteria of the query. Data are passed to printing function which prints out the data depending on the parameters.