Task documentation CLS: C++ Classes in Python 3 for IPP 2016/2017
Name and surname: Martin Ivančo
Login: xivanc03

Task overview

The task was to create a Python 3 script that analyses given code in simplified C++11 syntax. The script creates class dependency tree or details about chosen class in XML format.

Implementation parts

Argument processing

The script begins by parsing arguments. If the arguments are valid, argument variables are set accordingly. In case of invalid arguments or an invalid combination of them, the script ends with an error. This is done using auxiliary function `parse_arguments()`.

Getting input

If an input file has been specified, it is opened and loaded into string. If there was no input file specified, standard input is used. If the given file doesn't exist, the script ends with an error.

Parsing code

Input is passed to the function `parse_code()` which handles translating input code into objects contained in array `classes[]`. This array contains `Cla` objects which represent classes from the given input. Each one of these objects has got arrays of other objects representing methods and attributes of that class contained in them. The `parse_code()` function goes through the code and searches for classes. When it finds one, it immediately passes this part of the input to function `parse_class()`.

Parsing class

The `parse_class()` function goes through the whole class definition code, creating a new Cla object and adding appropriate inheritances, attributes and methods to it. To do this, it uses other helpful functions. When its finished, the `Cla` object is returned and later added to the `classes[]` array in `parse_code()` function.

Creating XML class dependency tree

According to the given arguments, either class dependency tree or class details in XML format are created. In case of class dependency tree, the function `make_xml_tree()` is called. This function goes through all `Cla` objects in `classes[]` array and adds them to the appropriate place in the XML tree. The resulting XML is returned.

Creating XML class details

If there has been a class specified, the `class_details_xml()` function is called for it once. If no class has been specified, this function is called multiple times, for each element of `classes[]` array once, while the returned XML is always appended to the resulting XML. The `class_details_xml()` function goes through all information available in the `Cla` object corresponding to the given name of class and transforms it to XML format.

Making pretty XML

If the `–pretty-xml` argument is specified, the script should output XML in a structured form, where each child is indented according to users request. If this argument is not

specified, the task leaves the form free, but my script still produces XML in a “pretty” form with default indentation of 4 spaces. This is easily done using `xml.dom.minidom` function `toprettyxml()`.

Exporting output

If an output file has been specified, it is opened and the resulting XML is written to it. In case there was no output file given, standard output is used.

Conclusion

Although I never worked with Python before, I was able to adapt quite quickly to the unusual syntax sensible on indentation. This project helped me learn the basics of this language, as well as some extra knowledge regarding regular expression searching and XML processing. Overall, this project was very useful.