Python Project 1
OpenMath support in Python

CS2006, 2015/16

Due date: Tuesday 8th March (week 7), 21:00
16.7%of Overall Mark for the Module

Overview

The objective of this project is to implement a basic support of the OpenMath encoding in Python 3.

OpenMath (`http://www.openmath.org/`) is an extensible standard for representing mathematical objects according to their semantics. It may be used to store and exchange mathematical data between different applications in a variety of possible scenarios, for example, to deal with computational problems that can not be solved within a single software system.

By completing this project, you should become comfortable with the basic use of Python, distributing the code of the Python project across multiple files, manipulating Python data types and working with XML parsers.

Preparation

In the `Practicals/Python/Project1/Code` directory you will find some code that may be used as a starting point:

- `omparse.py` containing the initial implementation of OpenMath parser, which supports only integers and lists.
- `omput.py` containing the initial implementation of `OMobject` function which produces the OpenMath encoding for its argument (also works only for integers and lists)
- `openmath.py` containing some higher level functionality to parse strings and files containing OpenMath objects and to print or return a string with the OpenMath encoding of the given object.
- `demo.py` which may be used for an interactive demo: open the terminal in the directory with these files, start Python interpreter, and then copy and paste its commands into the Python session.
- `tst` directory with a collection of XML files containing OpenMath objects to work with.

This Python code uses the ElementTree XML parser (`https://docs.python.org/3.4/library/xml.etree.elementtree.html`). There are other XML parsers for Python available, so you should feel free to use any of them, provided your implementation satisfies the requirements described below.

Basic Requirements

The minimum requirement for this project is to implement support for the following basic OpenMath objects:

- integers (see `tst/integer.xml`)
- IEEE floats (see `tst/float.xml`)
- strings (see `tst/string.xml`)

as documented in Sections 2.1.1 and 3.1.1 of the OpenMath standard (`http://www.openmath.org/standard/om20-2004-06-30/omstd20html-0.xml`).

This support should be bi-directional: it should be possible to convert an OpenMath XML representation of a string, float or integer into a Python variable and vice versa.

The files `omparse.py` and `omput.py` already implement support for integers, so you may proceed with adding support for floats and strings by analogy. Alternatively, you may use a different XML parser for Python, and then implement support for all three kinds of objects yourself.

For the efficient exchange of OpenMath-encoded data, one should be able to operate with higher-order objects assembled from basic OpenMath objects: matrices with integer or floating point entries, lists of strings, nested lists with arbitrary nesting patterns, etc.

As you may see, it is crucial to be able to handle lists, and here the OpenMath standard defines encoding for lists using the `list1.list` symbol, documented in `http://www.openmath.org/cd/list1.xhtml#list`. You may find some examples of OpenMath lists in `tst/list.xml` and `tst/listnested.xml` and observe there the usage of two other OpenMath constructions: OpenMath symbols (OMS) and OpenMath applications (OMA).

Informally, a symbol may describe a mathematical object itself (e.g. a boolean value, infinity, or the set of all rational numbers) or correspond to a function to construct an output from its input (e.g. to calculate factorial of an integer or just to return a list of its arguments). In the latter case, OpenMath application should be used as a construction which encodes the "function call".

OpenMath symbols are organised into so called *content dictionaries*. Content dictionaries (CDs) are collected at `http://www.openmath.org/cd/` and document the exact meaning of OpenMath *symbols*, permitting multiple systems to "speak the common language". Any reasonable OpenMath implementation should be designed in a way that makes it extendable with adding support for input and/or output of more OpenMath symbols if there will be a need in the future (perhaps even including "private" content dictionaries not listed on the official OpenMath website).

An example of such extendable framework for parsing OpenMath symbols (OMS) and OpenMath applications (OMA) is given in the `omparse.py` file (see `ParseOMelementHandler` and `omdicts` dictionaries and related code). It provides support for input and output in OpenMath for lists whose entries are integers or nested lists of integers (with arbitrary nesting patterns).

Now we are ready to introduce the next basic requirement: implementing support for a selection of OpenMath *content dictionaries*, either by extending the initial implementation described above, or by re-implementing it using a different XML parser for Python.

You should implement support for the following symbols:

- `logic1.true`, `logic1.false` (input/output)
- `nums1.rational` (input)
- `complex1.complex_cartesian` (input/output)
- `interval1.integer_interval` (input)
- `linalg2.matrix`, `linalg2.matrixrow` (input)

where, for example, `logic1.true` means the symbol `true` from the `logic1` CD (so you may find it in the list of all CDs at `http://www.openmath.org/cdnames.html` to see the documentation).

To demonstrate that your implementation works, you should write a test that will parse appropriate `.xml` files from the `tst` subdirectory. You may create additional `.xml` files, if necessary. For the cases

when conversion between Python and OpenMath is supported in both directions, a useful test is to produce the OpenMath encoding for an object, then parse it and check that the result of the parsing is equal to the original object.

Additional Requirements

Note: It is strongly recommended to ensure that you have completed the Basic Requirements and have something to submit before you attempt to deal with the Additional Requirements.

In order to achieve a grade higher than 13, you should implement some of the additional requirements below. In particular, for a grade higher than 17, you should implement all of the requirements marked as **Easy** and **Medium**. You should not be limited by these and should feel free to implement any other feature that you consider useful (please make sure that it will be described in the report!)

- **Easy:** Implement pretty-printing for OpenMath objects.
- **Easy:** Develop and implement a content dictionary to encode Python dictionaries. You may assume that they will be mapped to records in other programming languages. Your report should include a short specification describing the meaning and use of OpenMath symbol(s) to represent Python dictionaries. Your implementation should work for `key:value` pairs where the `value` is any object that may be encoded in OpenMath using your project. It should also give a meaningful error message if conversion of the `value` to OpenMath is not supported.
- **Medium:** Implement support for arithmetical operations from `arith1` CD.
- **Medium:** Implement support for `integer1.factorial`.
- **Medium to Hard:** Write a script that reads an OpenMath object, performs some meaningful calculation and then writes the result in OpenMath.
- **Medium to Hard:** Implement some other non-trivial OpenMath symbol(s) of your choice.
- **Medium to Hard:** Implement functionality to output OpenMath objects using prescribed content dictionaries. For example, one could try to output matrices using `linalg2.matrix` and `linalg2.matrixrow` instead of using lists.
- **Hard:** Implement support for OpenMath attributions (see OpenMath standard, Sections 2.1 and 3.1).
- **Hard:** Implement support for OpenMath errors (see OpenMath standard, Sections 2.1 and 3.1).
- **Hard:** Using Python's socket module (`https://docs.python.org/3.4/library/socket.html`), implement client–server communication between two Python instances which will use OpenMath to send remote procedure calls and get back the results. Use exception handling to catch and transmit an error to the client using OpenMath errors.

Deliverables

Hand in via MMS, by the deadline of 9pm on Tuesday of Week 7, a single `.zip` or `.tar.gz` file containing two top level subdirectories called `Code` and `Report`, as follows:

The `Code` directory should contain your group's code, and should be the same for everyone in the group. There may be further subdirectories if you wish, containing the source code and data in well-commented `.py` and `.xml` files. Everything that is needed to run your application should be in that directory or part of the standard Python installation: there should be no dependencies on external libraries. The marker should be able to run your code from the command line of a bash shell or from the Python interpreter, and you should document how to run it.

The `Report` directory should contain an individual report (of around 1500–2000 words), in PDF format, describing your design and implementation and any difficulties you encountered. In particular, it should include:

- A summary of the functionality of your program indicating the level of completeness with respect to the Basic Requirements, and any Additional Requirements.
- Any known problems with your code, where you do not meet the Basic Requirements or any Additional Requirements you attempted.
- Any specific problems you encountered which you were able to solve, and how you solved them.
- An accurate summary of provenance, i.e. stating which files or code fragments were:
  1. written by you;
  2. modified by you from the source files provided for this assignment;
  3. sourced from elsewhere and who wrote them.
- A description of your own contribution to the group work.

Marking Guidelines

This practical will be marked according to the guidelines at `https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html`. To give an idea of how the guidelines will be applied to this project:

- The most simple solution which supports input and output of OpenMath integers, floats and strings, accompanied by a report, should get you a grade 7.
- Completing the basic requirements with a well-commented/documented, fully-working solution, accompanied by a clear and informative report, should get you a grade 13. The report should address all the Basic Requirements, and should make clear which of these you have completed.
- To achieve a grade higher than 17 you will need to implement the Easy and Medium additional requirements listed above, with well-commented and documented code, accompanied by a clear and informative report.

Finally, remember that:

- Standard lateness penalties apply as outlined in the student handbook at `https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalt`
- Guidelines for good academic practice are outlined in the student handbook at `https://info.cs.st-andrews.ac.uk/student-handbook/academic/gap.html`

Finally

The project is open-ended, so feel free to discuss your own agenda in addition to implementing Easy and Medium requirements in order to score a high mark. Ideas may include, but are not limited to: developing interesting applications using OpenMath, comparing different XML libraries available in Python and their suitability for working with OpenMath, implementing a framework for distributed parallel calculations, etc. Finally, please let me, a tutor or a demonstrator know if you have any questions or problems!