OpenMath support in Python

Files submitted

omput.py	Module contains methods for converting python objects into OM strings
omparse.py	Module contains methods for converting OM strings into Python objects
arith_func.py	Module contains the pure mathematical functions associated with operations from arith1 cd and factorial from integer1
OpenMath.py	Module contains methods for two way processing between OM files (or strings) and Python objects
omclient.py	Network client that connects to a server and sends file contents.
omserver.py	Multithreaded OpenMath server that evaluates OpenMath strings.
test.py	Module contains tests that involve processing OM files and displaying (+saving) the result in OM format.
tst	directory contains the test files used by **test** method in **test.py**
tst_out	directory contains the result of processing files from **tst** directory as OpenMath xml files

Level of completion

cd name	object name	Level of support
		green = bi-directional, where appropriate
		(i.e. bi-directional support for **sum** would not make sense)
		yellow = other

list1	list	🟩
logic1	true, false	🟩
nums1	rational, infinity	🟩
complex1	complex cartesian	🟩
interval1	integer interval	🟩
linalg2	matrix, matrix row	🟩
arith1	all objects	🟩
integer1	factorial	🟩
error	all objects	🟩
fns1	lambda	🟨 **input only** supports: ○ single variable functions used to process **sum** and **product** applications ○ multiple variable functions ○ may output nested lambda functions. Such a single variable function can be then flattened using **flatten()**

>> Basic requirements completed

>> Easy extensions:

- ✓ Pretty printing

>> Medium extensions:

- ✓ All completed

>> Medium – Hard extensions:

- ✓ script that reads an OpenMath object, performs some meaningful calculation and then writes the result in OpenMath:
  - **process()** function returns result of processing in OpenMath and in python formats this result is then written to a file in **tst_out** directory
- ✓ bi-directional support for matrix and matrix row using appropriate dictionary

>> Hard extensions:

- ✓ Support for all objects in the **error** CD
- ✓ Multi-threaded client server communication between python instances

>> Additionally:

- ✓ Support for such errors as: file not found, invalid OpenMath string inside .xml file using the **ProgramErrorObj** class.
- ✓ Error handling for the networking part of the program
- ✓ Automated test function
  - Processes all files from **tst** directory and double converts the result to check that the expected bi-directional support has been provided for all objects
  - Displays the results for each file + the list of files that did not pass the test

Design & implementation

As a group, we have started working by making a list of all tasks mentioned in the Specification. Each of us implemented tasks of their choice, regularly updating our checklist with their contribution.

All the objects were implemented similarly to the **integer** and **list** objects whose implementation has already been provided.

Matrix class

- ✓ We decided to store the OpenMath matrix object as a list of lists in python.

- ✓ We created the Matrix and MatrixRow classes and defined the __eq__ method for them, which returns true if two matrices contain the same elements in corresponding rows.

- ✓ Defining these classes allowed us to implement bi-directional support for matrices. The OMelement() method in **omput.py** is now able to identify a matrix by checking if the object is an instance of the respective class.

Error classes

- ✓ We implemented two separate classes for errors of type:
  - OpenMath error (defined by the **error** CD)
  - program error – occurs if an OpenMath .xml file or string could not be processed correctly. The reasons may be:
    - file not found
    - unknown tag inside OpenMath string
    - nothing to process inside the OpenMath string – no actual object has been provided
    - no implementation has been provided for encoding a specific python object into OpenMath

- ✓ Bi-directional support has been provided for OpenMath errors: the program can read an error, but also output it using the appropriate encoding if the error occurs while processing another OpenMath string: e.g. dictionary not found, symbol not found, symbol found but not supported

- ✓ The three types of OpenMath errors mentioned above occur when parsing an OMS element, so we decided to include the appropriate checks inside the **ParseOMS()** method found in **omparse.py**

- ✓ Both error classes store the name of the error (or type, respectively) and the context in which it occurred.
  - For OMErrorObj the context is the node that could not be parsed
  - For ProgramErrorObj the context is the OpenMath element tag, the python object that could not be encoded or the name of the file that could not be processed.

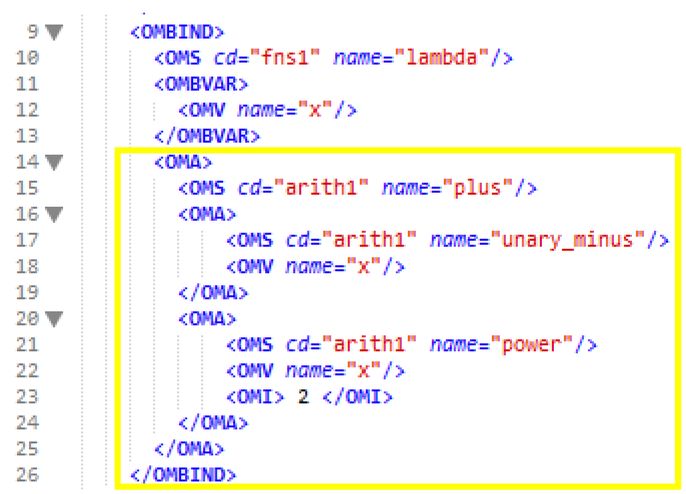
- ✓ Objects of type ProgramErrorObj are used to improve the user – program interaction. When such an error occurs, we display its type and the context in which it occurred. This feature is used by the test method when displaying the test results.

Implementing support for **lambda** object:

One of the challenging tasks was to implement support for lambda functions in order to implement the **sum** and **product** OpenMath objects from the **arith1** content dictionary.

We had to come up with a way of linking the variables defined within the <OMBVAR> element and the actual function body, preceded by the <OMS cd='fns1' name='lambda'> tag.

This has been achieved by using a content dictionary: **env{}** that represents the variable environment of the function. It stores the name of the variable and a pointer to it as a key:value pair.
Thus, we create the link between the variable names from <OMV name='x'> and the actual variables as we compose the lambda function:



```
<OMBIND>
  <OMS cd="fns1" name="lambda"/>
  <OMBVAR>
    <OMV name="x"/>
  </OMBVAR>
  <OMA>
    <OMS cd="arith1" name="plus"/>
    <OMA>
        <OMS cd="arith1" name="unary_minus"/>
        <OMV name="x"/>
    </OMA>
    <OMA>
        <OMS cd="arith1" name="power"/>
        <OMV name="x"/>
        <OMI> 2 </OMI>
    </OMA>
  </OMA>
</OMBIND>
```

When we encounter the variable definition we create a reference to this variable by defining a lambda function.

```
return lambda x: eval(vs, body,updateEnv(varname,x,env))
```

As we compose the function, we update the variable environment (**updateEnv()** method) with the name – pointer relation.

Each operation (OMS application of an arithmetic operation) within the function body (yellow) results in a new lambda function.

When there are no more variables defined inside <OMBVAR> element, we start recursively parsing the body, passing the **env{}** dictionary along. When we encounter an OMS application of any of the operations defined by **arith1**, we return the pure mathematical function associated with the corresponding operation.

- ✓ These pure functions are found in **arith_func.py**
- ✓ The arguments passed to the pure functions are either actual values or pointers to variables.
- ✓ We retrieve the pointers from **env{}**

The eval function makes use of the **evalBody()** and **updateEnv()** functions to achieve the construction of the final function. The function returned by **eval()** may be nested, so before passing it to the implementations of **sum** and **product** OpenMath operations, we need to flatten it.

flatten() takes a nested lambda function with a single variable and recursively reduces it to a 'flat' function, which can then be used by **sum** and **product** implementations.

Design of the network part

- We implemented a simple multi-threaded server (has to be run first)

- The client can specify a file to be processed. It is then sent to the server.

- Maximum size of the file is 4096 bytes.

- The server uses processing functions from the openmath.py module to evaluate the openmath string and sends the result back to the client as a pretty string also in openmath format.

- The user then sees the result of processing the file at the console and can choose to save it locally, specifying a name for the result file to be saved.
- The omclient.py module is completely independent from modules involved in processing the openmath file. Thus, all the processing is done on the server side.
- This feature allows anyone to easily connect to our server and use the functionality of our program.

Test program

In order to demonstrate that our program provides bidirectional support for the objects listed in the **Level of Completion** section, we have implemented an automated test method found inside **test.py**.
We compare result of **process()** function from **OpenMath.py** with the outcome of it's double convertion : to OpenMath and back to Python in order to decide the outcome of each test.

See next section for more details.

Testing

Note: As a group, we decided to each test a part of the program, create a test table or take screenshots for that part and share it with other members to be used in our reports. ***Table 1 and Table 2*** were created by me.

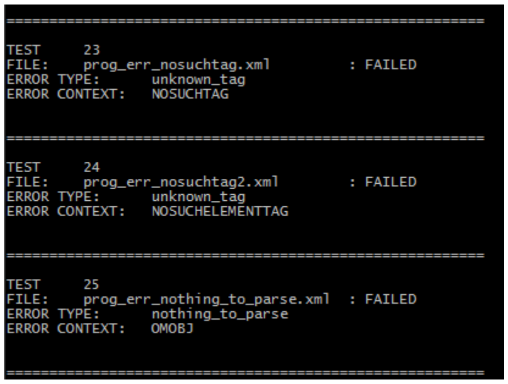
test.py contains the method for automatically testing the program using the files from tst directory.
The main assertion is: `result.python== ParseOMstring(OMstring(result.python))`

, where **result** is the PythonOM package returned by the OpenMath **process()** function. It contains the result of processing file **x** from **tst** directory.
The result package contains the result as:

- A python object - accessed using **result.python**
- An OpenMath pretty printed string accessed using **result.OpenMath**

We double convert the **result.python** object : to OpenMath and back to python representation and check that it is the same as the original result.

The result of this comparison determines whether the tests have passed or not. The fact that all tests in the current **tst** directory pass (except for the prog_err_ files) shows that the respective objects are correctly handled by the program and bi-directional support has been provided for them.



```
==============================================================

TEST     23
FILE:    prog_err_nosuchtag.xml          : FAILED
ERROR TYPE:       unknown_tag
ERROR CONTEXT:    NOSUCHTAG


==============================================================

TEST     24
FILE:    prog_err_nosuchtag2.xml         : FAILED
ERROR TYPE:       unknown_tag
ERROR CONTEXT:    NOSUCHELEMENTTAG


==============================================================

TEST     25
FILE:    prog_err_nothing_to_parse.xml   : FAILED
ERROR TYPE:       nothing_to_parse
ERROR CONTEXT:    OMOBJ


==============================================================
```

Generally, all tests in the tst directory should always pass, providing that:

- tst contains only .xml files
- all xml files contain valid OpenMath strings

The prog_err_ files contain errors in the OpenMath encoding and an appropriate ProgramErrorObj is returned when they are processed. The results of processing the prog_err files as well as the the valid OpenMath files containing mathematical objects and expressions are presented in the table below:

Table 1

File	Comment	Actual result
prog_err_nosuchtag	unknown tag wrapping the OpenMath object	appropriate error raised
prog_err_nosuchtag2	unknown tag inside an OpenMath object	appropriate error raised
prog_err_nothing_to_process	OpenMath object does not contain anything except for the OMOBJ tag	appropriate error raised

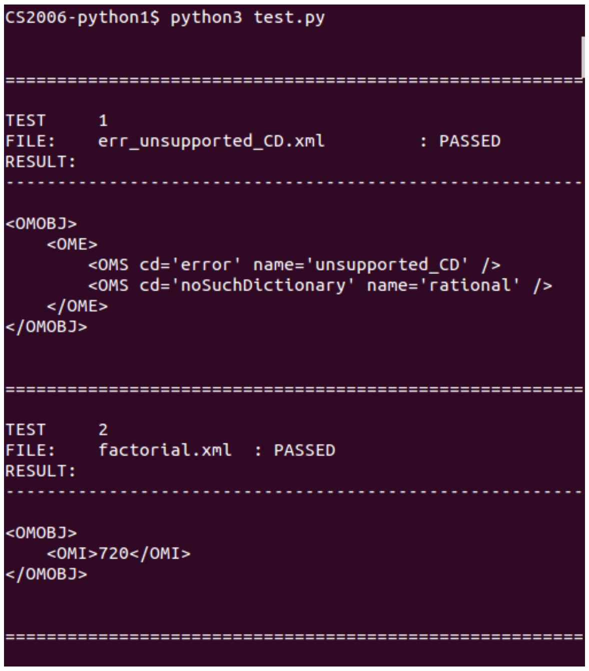
Table 2 - testing that the result of processing openmath files is correct from the mathematical point of view

File tested	Mathematical expression contained	Result of processing (as a Python object) green = test passed
abs.xml	abs (-15-(50-20)) = 45	45
bool.xml	false	false
complex.xml	2/3 + 5/4 j	(Fraction(2,3)+Fraction(5/4)j)
divide.xml	18/0.5=36	36
err_readingError.xml	reading an OpenMath error object	error object unchanged
err_unexpected_symbol.xml	invalid name for oms	appropriate error object
err_unsupported_CD.xml	invalid name for cd	appropriate error object
err_unsupported_symbol.xml	oms not supported	appropriate error object
factorial.xml	(10-7)!! = 3!! = 6!!= 720	720
float.xml	{0,1.0,0.5,-1.0,19487171.0..}	python list with the corresponding float numbers
gcd_lcm.xml	gcd(10,15) + lcm(10,15) = 5+30=35	35
integer.xml	42	42
interval.xml	[1..10]	python list: [1,2,3…10]
list.xml	[41, true, false, 43]	same list in python
listnested.xml	[1,2,[3,4,5]]	same list in python
matrix.xml	(1,2,3) (42,5,6) (0,-1,100)	Matrix object will rows = list of the following MatrixRow objects [1,2,3], [42,5,6],[0,-1,100]
plus.xml	-15+18=3	3
plus_minus.xml	-15+50-20=15	15
plus_minus_unary_minus.xml	-75+50-(-20)=-5	-5
power.xml	(10^-3)^(-2) = 1 mln	float 1000000.0
power_root.xml	((3rd root of 1000)^ 2) ^ -1 = 0.01	float 0.01
product.xml	product of x in [1..3] for f(x) = sqrt(x) = sqrt(6)	float 2.449…
range.xml	[-10..10]	python list of all values in this range
rational.xml	[1,1/2]	[1,Fraction(1,2)]
string.xml	This is a string	same as python string
sum.xml	sum of x in [1..3] for f(x) = -x+2x = 6	6
sum2.xml	sum of x in [1..3] for f(x)=x^2-x = 8	8
times.xml	8*1/2=4	4

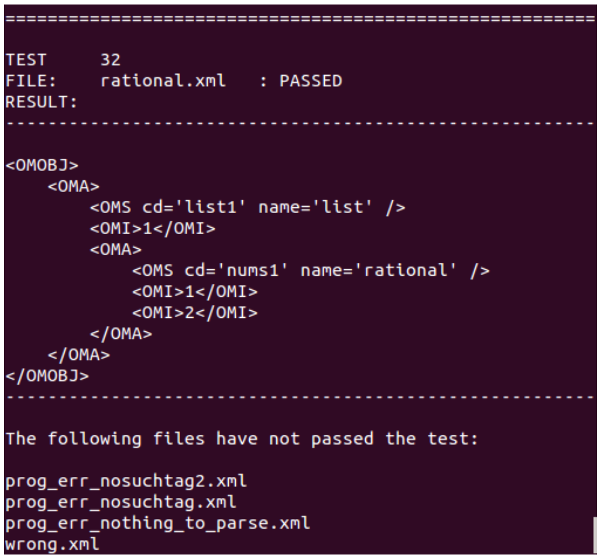
Table 3 - testing the networking part of the program

Test description	Expected result	Screenshot	Result green = passed
Normal file transmission of valid 'tst/integer.xml'	Get correct integer result	normal_network.png	🟩
Normal file transmission of erroneous 'tst/err_unexpected_symbol.xml' and save result	Get correct error result and save file	error_network.png	🟩
Quitting server with ":q"	Server quits	quit_server.png	🟩
Connecting to offline server	Cannot connect to server	offline_server.png	🟩
Sending non-OpenMath file	Show appropriate error message	non_xml_file.png	🟩
Sending non-existent file and quitting client	Does not send file	invalidfile_client.png	🟩
Connecting to one server with two clients	Server handles both clients' requests	multiclient.png	🟩

Testing continued– screenshots from **test()** method

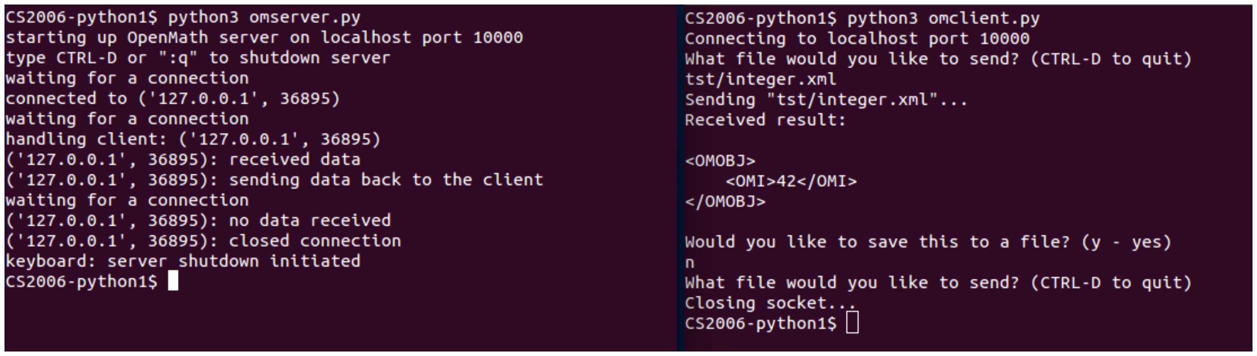


```
CS2006-python1$ python3 test.py


=========================================================

TEST     1
FILE:    err_unsupported_CD.xml         : PASSED
RESULT:
---------------------------------------------------------

<OMOBJ>
    <OME>
        <OMS cd='error' name='unsupported_CD' />
        <OMS cd='noSuchDictionary' name='rational' />
    </OME>
</OMOBJ>


=========================================================

TEST     2
FILE:    factorial.xml   : PASSED
RESULT:
---------------------------------------------------------

<OMOBJ>
    <OMI>720</OMI>
</OMOBJ>


=========================================================
```



```
=========================================================

TEST     32
FILE:    rational.xml    : PASSED
RESULT:
---------------------------------------------------------

<OMOBJ>
    <OMA>
        <OMS cd='list1' name='list' />
        <OMI>1</OMI>
        <OMA>
            <OMS cd='nums1' name='rational' />
            <OMI>1</OMI>
            <OMI>2</OMI>
        </OMA>
    </OMA>
</OMOBJ>
---------------------------------------------------------

The following files have not passed the test:

prog_err_nosuchtag2.xml
prog_err_nosuchtag.xml
prog_err_nothing_to_parse.xml
wrong.xml
```

Testing continued– screenshots for Table 3

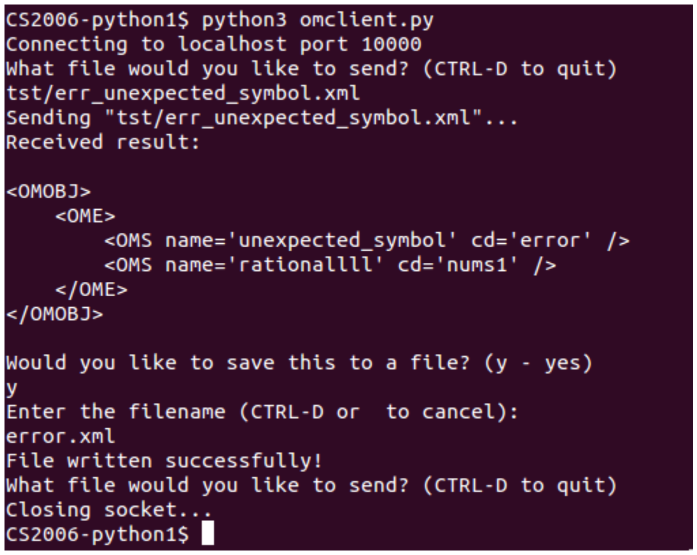


```
CS2006-python1$ python3 omserver.py
starting up OpenMath server on localhost port 10000
type CTRL-D or ":q" to shutdown server
waiting for a connection
connected to ('127.0.0.1', 36895)
waiting for a connection
handling client: ('127.0.0.1', 36895)
('127.0.0.1', 36895): received data
('127.0.0.1', 36895): sending data back to the client
waiting for a connection
('127.0.0.1', 36895): no data received
('127.0.0.1', 36895): closed connection
keyboard: server shutdown initiated
CS2006-python1$
```

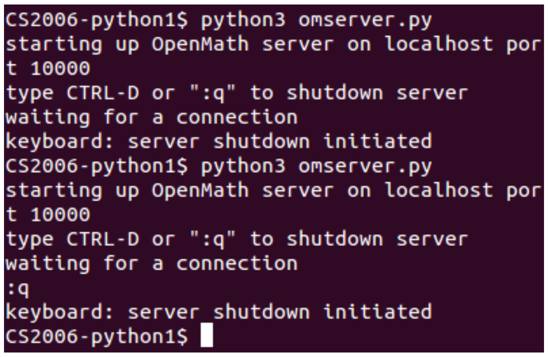
```
CS2006-python1$ python3 omclient.py
Connecting to localhost port 10000
What file would you like to send? (CTRL-D to quit)
tst/integer.xml
Sending "tst/integer.xml"...
Received result:

<OMOBJ>
    <OMI>42</OMI>
</OMOBJ>

Would you like to save this to a file? (y - yes)
n
What file would you like to send? (CTRL-D to quit)
Closing socket...
CS2006-python1$
```

Above: normal_network.png

Below: error_network.png

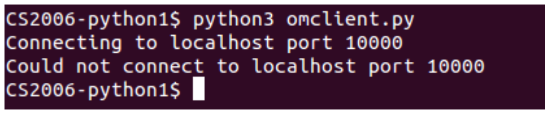


```
CS2006-python1$ python3 omclient.py
Connecting to localhost port 10000
What file would you like to send? (CTRL-D to quit)
tst/err_unexpected_symbol.xml
Sending "tst/err_unexpected_symbol.xml"...
Received result:

<OMOBJ>
    <OME>
        <OMS name='unexpected_symbol' cd='error' />
        <OMS name='rationallll' cd='nums1' />
    </OME>
</OMOBJ>

Would you like to save this to a file? (y - yes)
y
Enter the filename (CTRL-D or  to cancel):
error.xml
File written successfully!
What file would you like to send? (CTRL-D to quit)
Closing socket...
CS2006-python1$
```



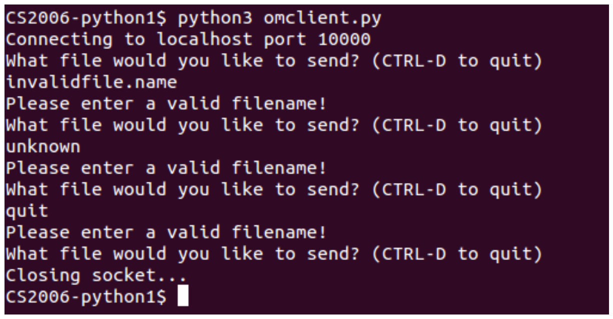
```
CS2006-python1$ python3 omserver.py
starting up OpenMath server on localhost por
t 10000
type CTRL-D or ":q" to shutdown server
waiting for a connection
keyboard: server shutdown initiated
CS2006-python1$ python3 omserver.py
starting up OpenMath server on localhost por
t 10000
type CTRL-D or ":q" to shutdown server
waiting for a connection
:q
keyboard: server shutdown initiated
CS2006-python1$
```

Above: quit_server.png
Below: offline_server.png



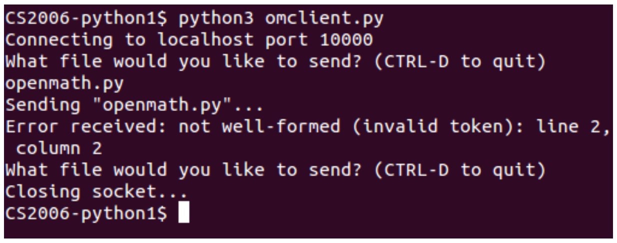
```
CS2006-python1$ python3 omclient.py
Connecting to localhost port 10000
Could not connect to localhost port 10000
CS2006-python1$
```

Below: invalidfile_client.png



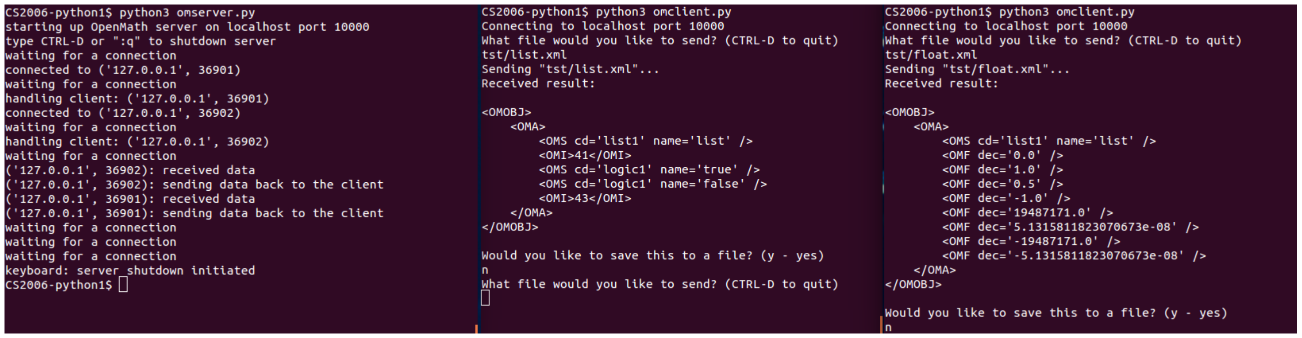
```
CS2006-python1$ python3 omclient.py
Connecting to localhost port 10000
What file would you like to send? (CTRL-D to quit)
invalidfile.name
Please enter a valid filename!
What file would you like to send? (CTRL-D to quit)
unknown
Please enter a valid filename!
What file would you like to send? (CTRL-D to quit)
quit
Please enter a valid filename!
What file would you like to send? (CTRL-D to quit)
Closing socket...
CS2006-python1$
```

Below: non_xml_file.png



```
CS2006-python1$ python3 omclient.py
Connecting to localhost port 10000
What file would you like to send? (CTRL-D to quit)
openmath.py
Sending "openmath.py"...
Error received: not well-formed (invalid token): line 2,
 column 2
What file would you like to send? (CTRL-D to quit)
Closing socket...
CS2006-python1$
```

Below: multiclient.png



```
CS2006-python1$ python3 omserver.py
starting up OpenMath server on localhost port 10000
type CTRL-D or ":q" to shutdown server
waiting for a connection
connected to ('127.0.0.1', 36901)
waiting for a connection
handling client: ('127.0.0.1', 36901)
connected to ('127.0.0.1', 36902)
waiting for a connection
handling client: ('127.0.0.1', 36902)
waiting for a connection
('127.0.0.1', 36902): received data
('127.0.0.1', 36902): sending data back to the client
('127.0.0.1', 36901): received data
('127.0.0.1', 36901): sending data back to the client
waiting for a connection
waiting for a connection
waiting for a connection
keyboard: server shutdown initiated
CS2006-python1$
```

```
CS2006-python1$ python3 omclient.py
Connecting to localhost port 10000
What file would you like to send? (CTRL-D to quit)
tst/list.xml
Sending "tst/list.xml"...
Received result:

<OMOBJ>
    <OMA>
        <OMS cd='list1' name='list' />
        <OMI>41</OMI>
        <OMS cd='logic1' name='true' />
        <OMS cd='logic1' name='false' />
        <OMI>43</OMI>
    </OMA>
</OMOBJ>

Would you like to save this to a file? (y - yes)
n
What file would you like to send? (CTRL-D to quit)
```

```
CS2006-python1$ python3 omclient.py
Connecting to localhost port 10000
What file would you like to send? (CTRL-D to quit)
tst/float.xml
Sending "tst/float.xml"...
Received result:

<OMOBJ>
    <OMA>
        <OMS cd='list1' name='list' />
        <OMF dec='0.0' />
        <OMF dec='1.0' />
        <OMF dec='0.5' />
        <OMF dec='-1.0' />
        <OMF dec='19487171.0' />
        <OMF dec='5.1315811823070673e-08' />
        <OMF dec='-19487171.0' />
        <OMF dec='-5.1315811823070673e-08' />
    </OMA>
</OMOBJ>

Would you like to save this to a file? (y - yes)
n
```

Summary of provenance

Code written by me

>> I have implemented bi-directional (except for lambda) support for the objects highlighted in orange in the table below:

cd name	object name	comment
list1	list	
logic1	true, false	
nums1	rational, infinity	
complex1	complex cartesian	
interval1	integer interval	
linalg2	matrix, matrix row	+ classes
arith1	all objects	
integer1	factorial	
error	all objects	
fns1	lambda	+auxiliary functions

>> Implemented the **arith_func.py** module
>> Implemented the **test.py** module + auxiliary functions used within the **test()** method.

Code modified from the source files

We used **omparse** and **omput** modules as a base for implementing bi-directional support for objects in the table above.

Openmath.py: We have added more functions related to processing openmath files + class defitinion for the PythonOM result package.

Demo.py has been replaced with **test.py**