COMP 304B – Object-Oriented Software Design
Assignment 1 – Unit testing and Python programming

Due date: Wednesday January 29, 2002 before 23:55

Practical information

- Team size == 2 (pair programming) !
- Each team submits only *one full solution*. Use the `index.html` template provided on the assignments page. Use **exactly** this format to specify names and IDs of the team members. The other team member *must* submit a single `index.html` file containing only the coordinates of both team members. This will allow us to put in grades for both team members in WebCT. Beware: after the submission deadline there is no way of adding the other team member's `index.html` file and thus no way of entering a grade !
- Your submission must be in the form of a simple HTML file (`index.html`) with explicit references to *all submitted files* as well as inline inclusion of images. See the general assignments page for an `index.html` template.
- The submission medium is WebCT.

Goals

This assignment will make you familiar with *unit testing* and object-oriented programming in *Python*.

You will test and implement classes to represent formulas used in spreadsheet cells. This is the very first step towards a simple spreadsheet application.

For testing, you will use the `unittest` framework in Python.

Your assignment should clearly present:

1. The tests corresponding to the requirements listed below.
2. An implementation of the design given below.
3. The results of the tests on your implementation.

Upload *all* source and result files to WebCT and provide links to them from your `index.html` file.

Design

In Figure 1, the design for this assignment is given in the form of a UML Class Diagram. You must test and implement all classes. The structure of the classes is given in the design. You must adhere *strictly* to the given design.

Note how the design specifies describe the abstract class `FNode` with its concrete realizations in the form of sub-classes. The `Operator` class is linked to the `FNode` class denoting the fact that an `Operator` node refers to its children. Actually, the design already specifies that the reference to the children will be through an attribute `_children` of type `List_Of_FNodes`.

Although a `Number` can be instantiated with either an Integer or a Float, internally, storage is as a `Float`. Furthermore, you are required to perform all arithmetic on `Floats` only. Using `Integer` arithmetic would complicate matters when

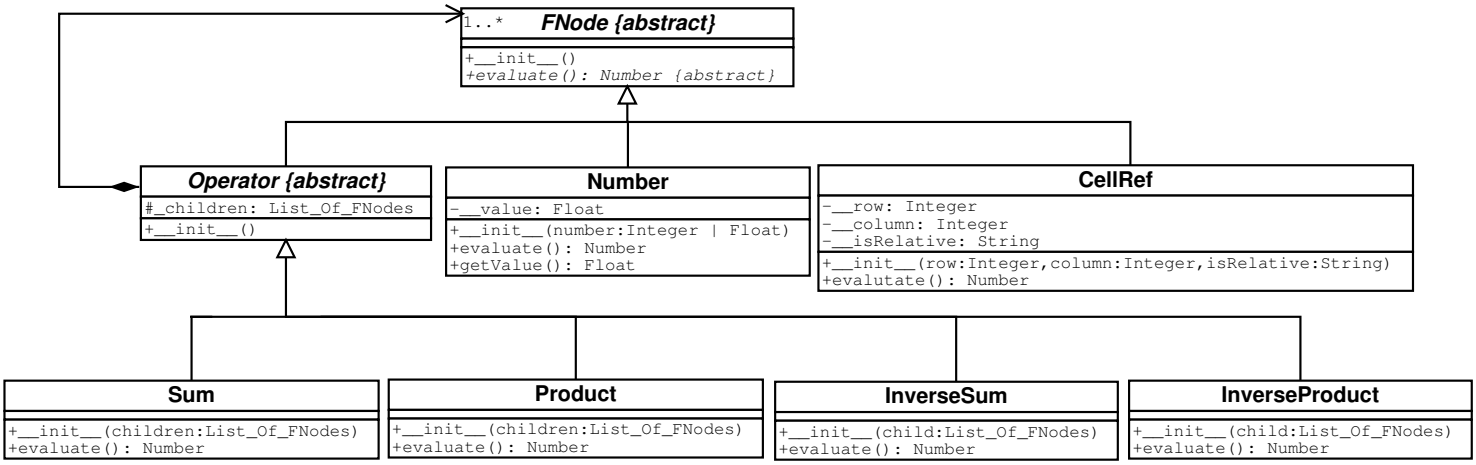

Figure 1: The spreadsheet cell formula node UML Class Diagram

dealing with division.
A side-effect of using `Floats` is that $1/1/x$ is usually *not* equal to x. This is due to the internal representation of floating point numbers and errors made during calculations. Hence, equality of two floating point numbers a and b will have to be tested by checking whether $|a-b| < \varepsilon$ with ε a tolerance.

You will implement the `evaluate()` method for all classes except `CellRef`. The latter will only be possible at the level of the overall spreadsheet (in a later assignment). Note how `evaluate()` does *not* return a `Float`, but in true object-oriented fashion, returns an object, instance of the `Number` class !

Requirements

1. Test for Success

1. Calling any concrete `evaluate` method must return an instance of class `Number`.
2. Build some tree data structures containing instances of classes `Operator` and `Number` (ignore class `CellRef` for now). Make sure "evaluating" the root node returns the expected result. For instance, if `a` is the root node of a tree representing the operation $2+4-5$, make sure that `a.evaluate.getValue()` returns 1.

2. Test for Failure

FNode:

- An attempt to instantiate this abstract class should raise a `NotImplementedError` exception.

Operator:

- An attempt to instantiate this abstract class should raise a `NotImplementedError` exception.

Number:

- Raise a `TypeError` if the argument passed to the constructor is not of the right type (`Integer` or `Float`).

CellRef:

- Raise a `TypeError` if the arguments passed to the constructor are not of the right type (`row` and `column` are `Integers`, `isRelative` is a `String`).

- Check that `refKind` is either `"RELATIVE"` or `"ABSOLUTE"`. If not, raise a `ValueError`.
- If `isRelative` equals `"ABSOLUTE"`, make sure that `row` and `column` are both non-negative. If not, raise a `ValueError`.

Sum and Product:

- Check that the parameter `children` is a `List`. If not, raise a `TypeError`.
- Check that the list contains at least two items. If not, raise a `TypeError`.
- Check that each item of that list is an instance (direct or not) of class `FNode`. If not, raise a `ValueError`.

InverseSum and InverseProduct:

- Check that the argument `child` is a `List`. If not, raise a `TypeError`.
- Check that the list contains a single item. If not, raise a `TypeError`.
- Check that the single item is an instance (direct or not) of class `FNode`. If not, raise a `ValueError`.

3. Test for Sanity

1. Do some sanity checks with classes `InverseSum` and `InverseProduct`. That is (for instance),

$$\begin{aligned} x+(-x) &= 0, \\ -(-x) &= x. \end{aligned}$$

2. Make sure the value returned by `Number.getValue` is consistent with the value stored in the attribute `Number.__value`.

3. Make sure the method `Number.evaluate` returns the caller object itself (use Python's `is` operator as in `a is b`).

References

- Extreme Programming (test centric, pair programming) http:www.extremeprogramming.org
- Unit testing in Pythontutorial (Chapter 6 of http://diveintopython.org)
- The original PyUnit site (unittesting is now included in the standard Python distribution) http://pyunit.sourceforge.net/.
- Dia, the gtk+ based diagram creation program used to draw the UML class diagram. http://www.lysator.liu.se/ alla/dia/ Dia is available in the SOCS labs on the FreeBSD machines.
- `design.dia` is the Dia source of our design.
- The `inheritance.py` example on the object-oriented software lecture page is a useful example of OO programming in Python.
- Have a look at the course's Frequently Asked Questions.