Extending the Python Interpreter - Laboratory Exercise

Peter Tröger

April 29, 2008

1 Introduction

It is expected that you know C language and the basic tools of open source software development (autoconf, make, gcc). It is also expected that you followed the Python lectures in the course.

The lab is intended to encourage your own ideas for extending the Python interpreter. Try to learn something about interpreter design by 'playing around', using this lab exercise as starting point.

The assignment contains several questions. You can answer them for yourself to check your true understanding of the underlying concepts.

2 The Task

In standard Python, the only way to exit the interactive interpreter is the Ctrl-D key combination (try it !). Most beginners instead expect an input like `exit` or `quit` to work. The Python people therefore included some static hacks in the Python interpreter, in order to print out an according message when users type in such a statement.

Within this laboratory exercise, we want to extend the Python language by a new statement, wich triggers the interpreter to finish execution:

```
tekpc552:python-svn troeger$ ./python.exe
Python 2.6a2+ (trunk:62564M, Apr 29 2008, 17:04:19)
[GCC 4.0.1 (Apple Inc. build 5465)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> bye
tekpc552:python-svn troeger$
```

We call this statement `bye`, in order to not conflict with the in-built `exit` and `quit` pseudo-statements. As all other Python statements (such as `print`, `raise`, or `pass`) it should work both in interactive and in scripting mode, simply by making it part of the language.

In the first version, the `bye` statement should take no parameters, similar to the `pass` statement.

3 Get, build and test the Python sources

- Go to `www.python.org` and obtain the Python sources. We recommend to download the TAR-file of the sources, instead of performing a (much slower) SVN checkout. Unpack the sources in your home directory (`tar xvfz Python-xxx.tgz`).
- Use the `./configure` command in the root directory of the sources to generate the Makefile on your machine.
- Use the `make` command to build Python. This might take a while, so you can already start to investigate the `Grammar/`, the `Parser/` and the `Python/` directories based on what you know from the lecture. Try to find the core parts of Python - the parser, the compiler and the interpreter. Which file contains the 'big switch statement' for the Python byte code execution ?
- Use the `make test` command to run the Python test suite. Try to understand some of the warning and error messages.
- Start your compiled Python and check if it works.

4 Extend the Python grammar

- Edit the file `Grammar/Grammar` to extend the Python language by a new `bye` statement. Look for the declaration of the `pass` statement to get an idea how the definition could work.
- Edit the file `Parser/python.asdl` accordingly to reflect the language extension in the AST structure elements.
- Re-build Python and try to understand how the following files have been regenerated according to the updated grammar:
  - `Python/Python-ast.c`
  - `Include/graminit.h`
  - `Include/Python-ast.h`

  You can do that by searching for the `bye` keyword in the source code.
- Check if the compiled Python interpreter already recognizes the new statement. Explain the resulting output when you use `bye` in interactive mode at this stage.

5 Implemented the language extension

- Edit `Python/ast.c` to support the new statement in the *concrete syntax tree* to *abstract syntrax tree* translation. You need to rely on the extended content of the `Include/Python-ast.h` file here. Take again the handling of the `pass` statement as example for the necessary extension.

- Re-build Python and check what happens now. Explain.
- Edit `Python/compile.c` to do something useful when the compiler finds our new statement in the process of AST parsing. Look where the `pass` statement is handled. For a first try, just let the compiler generate the byte code `PRINT_NEWLINE` that works without parameters. Re-build Python and check if it works.
- In order to let the language statement do what it is intended for, you need to raise a *SystemExit* exception (check `http://pyref.infogami.com/SystemExit`). We do this by adding the following statements to `Python/compile.c`, instead of putting out the `PRINT_NEWLINE` byte code:

  ```
  ADDOP_O(c, LOAD_GLOBAL, PyString_InternFromString("SystemExit"), names);
  ADDOP_I(c, RAISE_VARARGS, 1);
  ```

  Explain these statements.
- Re-build Python and see if it works.

6 Optional further tasks

- Print out a message before you raise the exception. Remember that printing has an own byte code in Python.
- Extend the `bye` statement to accept a numerical parameter with the exit code for the Python operating system process.
- Remove the in-build actions for `exit` and `quit` and trigger your own statement instead.
- Add a more complex statement to Python.
- Add a new operator to Python
  - Fancy operators (see `http://www.geocities.com/chrootstrap/adding_python_operators.html`)
  - Unary factorial
  - Binary over
- Add a new declarator in Python
  - `final` keyword as in Java
  - `abstract` keyword as in Java
- Optimize a typical Python activity by adding an own byte code for it.