“SWP”

A Generic Language Parser
(SWP == Semantic Whitespace Parser for want of a better name)

Michael Sparks
BBC R&D

Parse Anything

- Got bored of seeing “use Prothon”... “no”
- Hacking python to add a keyword whilst trivial wasn't trivial enough
- Got bored of seeing “use prothon's replacement”
- Thought it might be a fun thing to try
- Got bored of seeing “use the replacement for prothon's replacement”
- etc

Parse Anything

Parse this:

```
def displayResult(result,quiet):
   if not quiet:
      print "The result of parsing your program:"
      print result
      print
      if not result:
         print "Rule match/evaluation order"
         for rule in r:
            print "   ", rule
         end
      end
   else:
      if result is None:
         print "Parse failed"
      else:
         print "Success"
      end
   end
end
```

Parse Anything

Parse this:

```
#
# Sample logo like language using the parser
#

shape square:
   pen down
   repeat 4:
      forward 10
      rotate 90
   end
   pen up
end

repeat (360/5):
   square()
   rotate 5
end
```

Parse Anything

Parse this:

```
#
# Example based on defining grammars for L-Systems.
#
OBJECT tree L_SYSTEM:
  ROOT  G
  RULES:
      G -> T { G } { A G } { B G } { C G } (0.00 .. 0.15)
      G -> T { A B G } { B A G } { C A G } (0.15 .. 0.30)
      G -> T { A C G } { B B G } { C B G } (0.30 .. 0.45)
      G -> T { A A G } { B C G } { C C G } (0.45 .. 0.60)
      G -> T { A G } { C G }              (0.70 .. 0.80)
      G -> T { A G } { B G }              (0.80 .. 0.95)
      G -> T { A G }                      (0.95 .. 1.00)
      T -> T                              (0.00 .. 0.75)
  ENDRULES
ENDOBJECT
```

Parse Anything

Parse this:

```
#
# An SML-like language using this parser.
#
structure Stk = struct :
   exception EmptyStack_exception
   datatype 'x stack = EmptyStack | push of ('x * 'x stack)
   fun pop(push(x,y)) = y
   fun pop EmptyStack = raise EmptyStack_exception
   fun top(push(x,y)) = x
   fun top EmptyStack = raise EmptyStack_exception
end
```

Parse Anything, etc

```
EXPORT OBJECT person:
 PRIVATE:
   flat
   name, telephone
   address::PTR TO LONG
   telephone
 ENDATTRS
ENDOBJECT

PROC compare_address(address1::PTR TO LONG, address2::PTR TO LONG):
   #    Returns *TRUE* if the address2 exists _inside address1
   DEF result=TRUE, f
   FOR f:=0 TO 5:
      IF address2[f]:
         IF Not(((StrLen address2[f])==0) AND ((StrLen address1[f])==0)):
           #        The following line incorrectly(?) says that a
           #        NULL string does not exist inside a NULL string.
           #        The IF above corrects this
           result:=result AND ( ((InStr address1[f],address2[f])<>-1) OR ((StrLe
         ENDIF
      ENDIF
   ENDFOR
ENDPROC result
```

Parse This?!

```
OBJECT tree L_SYSTEM:
  ROOT  G
   structure Stk = struct :
      exception EmptyStack_exception
      datatype 'x stack = EmptyStack | push of ('x * 'x stack)
      shape square:
        repeat 4:
           forward 10
           rotate 90
        end
      end
   end
  RULES:
      G -> T { A G } { C G }        (0.70 .. 0.80)
      G -> T { A G } { B G }        (0.80 .. 0.95)
      G -> T { A G }                (0.95 .. 1.00)
  ENDRULES
ENDOBJECT
```

```
if (__name__ == "__main__"):
    import sys
    assign lexonly False
    assign trace False
    if sys.argv[1]:
       assign source open(sys.argv[1]).read()
    else:
       assign source "junk"
    end
end
```

Grammar

program -> block
block -> BLOCKSTART statement_list BLOCKEND
statement_list -> statement*
statement -> (expression | expression ASSIGNMENT expression |) EOL
expression -> oldexpression (COMMA expression)*
oldexpression -> (factor [factorlist] | factor INFIXOPERATOR expression)
factorlist -> factor* factor
factor -> (bracketedexpression | constructorexpression | NUMBER | STRING | I
| factor DOT dotexpression | factor trailer | factor trailertoo)
dotexpression -> (ID bracketedexpression | factor)
bracketedexpression -> BRA [expression] KET
constructorexpression -> BRA3 [expression] KET3
trailer -> BRA2 expression KET2
trailertoo -> COLON EOL block

Notes

- Just uses a slightly modified PLY
- All of the examples are parseable by the same parser – no changes to the lexer or parser.
- Just spits out a syntax tree
- Treats everything as a function

Everything's a function

- This is a function:

```
if bar(bibble=>baz):
    bla bla bla
    bingle
    bongle
else:
    babble babble
    this = bing
```

- Parsed as: Call function “if” with the arguments:
  bar(bibble=>baz), codeblock, “else”, codeblock, “endif”

Where...?

- http:///www.cerenity.org/SWP-0.0.0.tar.gz
- http://www.cerenity.org/SWP/
- I'd be curious to see someone put a lisp back end on it :-)