R"""A parser for the FC2 common format strings (expressions)

Author : Anders Andersen
Created On : Wed Dec 02 12:46:31 1998
Last Modified By: Anders Andersen
Last Modified On: Fri Mar 5 11:52:11 1999
Status : Unknown, Use with caution!

"""

```
import re

re_exp_comm = re.compile(r'\s*,\s*')

token_reexp = [
    ("boolval", re.compile(r'(true|false)(?!\w)')),
    ("boolop",  re.compile(r'(and|\^|or|v)(?!\w)')),
    ("notop",   re.compile(r'(not|~)(?!\w)')),
    ("mesg",    re.compile(r'([a-zA-Z]\w*!)')),
    ("evnt",    re.compile(r'([a-zA-Z]\w*\?)')),
    ("name",    re.compile(r'([a-zA-Z]\w*(\?)?)')),
    ("number",  re.compile(r'([0-9]+(\.[0-9]+)?)')),
    ("assign",  re.compile(r'(:=)')),
    ("numop",   re.compile(r'(\+|-|\*|/)')),
    ("testop",  re.compile(r'(<>|<=|>=|<(?!=)|>(?!=)|=)')),
    ("leftbr",  re.compile(r'(\()')),
    ("rightbr", re.compile(r'(\))')),
    ("compose", re.compile(r'(\|\|)')),
    ("ws",      re.compile(r'(\s+)'))]

token_map = {
    'true': '1',
    'false': '0',
    ':=': '=',
    'and': ' and ',
    '^': ' and ',
    'or': ' or ',
    'v': ' or ',
    'not': 'not ',
    '~': 'not ',
    '=': '==',
    '<>': '!='}

def tokenizer(string):
    pos = 0; tokens = []; length = len(string)
    while pos < length:
        for (type, reexp) in token_reexp:
            if reexp.match(string[pos:]):
                match = reexp.match(string[pos:])
                end = pos + match.end()
                tokens.append((type, string[pos:end]))
                pos = end
                break
        else:
            raise FC2Exception, "Unable to tokenize %s (%d)" % (string, pos)
    return tokens
```

```
def detokenizer(tokens, prename=""):
    string = ""
    for (type, token) in tokens:
        if type == "name":
            string = string + prename + token
        elif type == "group":
            string = string + "(" + detokenizer(token, prename) + ")"
        elif type != "ws":
            try:
                string = string + token_map[token]
            except KeyError:
                string = string + token
    return string

def bracketgroup(tokens):
    level = 1
    pos = 0
    grp = []
    while pos < len(tokens):
        if tokens[pos][0] == "leftbr":
            level = level + 1
        if tokens[pos][0] == "rightbr":
            level = level - 1
        if level == 0:
            return (pos, grp)
        grp.append(tokens[pos])
        pos = pos + 1
    return (pos, grp)

def splittest(tokens, prename=""):
    grps = []
    numtok = 0
    while numtok < len(tokens):
        (type, token) = tokens[numtok]
        numtok = numtok + 1
        if type == "leftbr":
            (num, grp) = bracketgroup(tokens[numtok:])
            numtok = numtok + num + 1
            grps.append(("group", grp))
        else:
            grps.append((type, token))
    for optype in ["boolop", "testop"]:
        numtok = 0
        while numtok < len(grps):
            if grps[numtok][0] == optype:
                return (optype, (
                    detokenizer(grps[numtok:numtok+1], prename),
                    detokenizer(grps[:numtok], prename),
                    detokenizer(grps[numtok+1:], prename)))
            numtok = numtok + 1
    return ("const", detokenizer(grps))

def jointest(test):
    if test[0] == "const":
        return test[1]
    else:
        return test[1][1] + test[1][0] + test[1][2]
```

```
def isstmt(tokens):
    if len(tokens) > 2:
        if tokens[0][0] == "name" and tokens[1][0] == "assign":
            return 1
    return 0

def ismesg(tokens):
    if len(tokens) == 1:
        if tokens[0][0] == "mesg":
            return 1
    return 0

def isevent(tokens):
    if len(tokens) == 1:
        if tokens[0][0] == "evnt":
            return 1
    return 0

def isname(tokens):
    for (type, token) in tokens:
        if type not in ["name", "compose"]:
            return 0
    return 1

def string_parser(string, prename=""):
    tokens = tokenizer(string)
    if isstmt(tokens):
        return ("stmt", detokenizer(tokens, prename))
    elif ismesg(tokens):
        return ("mesg", tokens[0][1][:-1])      # Shortcut
    elif isevent(tokens):
        return ("evnt", tokens[0][1][:-1])      # Shortcut
    elif isname(tokens):
        return ("name", detokenizer(tokens, ""))
    else:
        return ("test", splittest(tokens, prename))

def split_string(expr, prename=""):
    R"""Split a string to a list of typed elements
```

This takes a FC2 string expressions and splits it to a list of `"name"`, `"mesg"`, `"stmt"` and `"test"` elements. A string like `"off,x<3,x:=x+1"` will be returned as the following Python list:

```
[("name", "off"), ("test", "x<3"), ("stmt", "x=x+1")]
```

Note that the tests and expressions are converted to the corresponding Python syntax.

```
    """
    str_list = []
    for substr in re_exp_comm.split(expr):
        str_list.append(string_parser(substr, prename))
    return str_list
```