```
"""Module to analyze Python source code; for syntax coloring tools.
■
Interface:■
    tags = fontify(pytext, searchfrom, searchto)■
■
The 'pytext' argument is a string containing Python source code.■
The (optional) arguments 'searchfrom' and 'searchto' may contain a slice in pytext. ■
The returned value is a list of tuples, formatted like this:■
    [('keyword', 0, 6, None), ('keyword', 11, 17, None), ('comment', 23, 53, None), etc. ]■
The tuple contents are always like this:■
    (tag, startindex, endindex, sublist)■
tag is one of 'keyword', 'string', 'comment' or 'identifier'■
sublist is not used, hence always None. ■
"""■
■
# Based on FontText.py by Mitchell S. Chapman,■
# which was modified by Zachary Roadhouse,■
# then un-Tk'd by Just van Rossum.■
# Many thanks for regular expression debugging & authoring are due to:■
#    Tim (the-incredib-ly y'rs) Peters and Cristian Tismer■
# So, who owns the copyright? ;-) How about this:■
# Copyright 1996-2001: ■
#    Mitchell S. Chapman,■
#    Zachary Roadhouse,■
#    Tim Peters,■
#    Just van Rossum■
■
__version__ = "0.4"■
■
import string■
import re■
■
# First a little helper, since I don't like to repeat things. (Tismer speaking)■
import string■
def replace(where, what, withs):■
    return string.join(string.split(where, what), withs)■
■
# This list of keywords is taken from ref/node13.html of the■
# Python 1.3 HTML documentation. ("access" is intentionally omitted.)■
keywordsList = [■
    "assert", "exec",■
    "del", "from", "lambda", "return",■
    "and", "elif", "global", "not", "try",■
    "break", "else", "if", "or", "while",■
    "class", "except", "import", "pass",■
    "continue", "finally", "in", "print",■
    "def", "for", "is", "raise", "yield"]■
■
# Build up a regular expression which will match anything■
# interesting, including multi-line triple-quoted strings.■
commentPat = r"#[^\n]*"■
■
pat = r"q[^\\q\n]*(\\[\000-\377][^\\q\n]*)*q"■
quotePat = replace(pat, "q", "'") + "|" + replace(pat, 'q', '"')■
■
# Way to go, Tim!■
pat = r"""■
    qqq■
    [^\\q]*■
    (■
        (    \\[\000-\377]■
        |    q■
            (    \\[\000-\377]■
            |    [^\q]■
            |    q■
                (    \\[\000-\377]■
                |    [^\\q]■
                )■
            )■
        )■
```

```
        [^\\q]*
    )*
    qqq
"""
pat = string.join(string.split(pat), '')    # get rid of whitespace
tripleQuotePat = replace(pat, "q", "'") + "|" + replace(pat, 'q', '"')

# Build up a regular expression which matches all and only
# Python keywords. This will let us skip the uninteresting
# identifier references.
# nonKeyPat identifies characters which may legally precede
# a keyword pattern.
nonKeyPat = r"(^|[^a-zA-Z0-9_.\"'])"

keyPat = nonKeyPat + "(" + "|".join(keywordsList) + ")" + nonKeyPat

matchPat = commentPat + "|" + keyPat + "|" + tripleQuotePat + "|" + quotePat
matchRE = re.compile(matchPat)

idKeyPat = "[ \t]*[A-Za-z_][A-Za-z_0-9.]*"    # Ident w. leading whitespace.
idRE = re.compile(idKeyPat)


def fontify(pytext, searchfrom = 0, searchto = None):
    if searchto is None:
        searchto = len(pytext)
    # Cache a few attributes for quicker reference.
    search = matchRE.search
    idSearch = idRE.search

    tags = []
    tags_append = tags.append
    commentTag = 'comment'
    stringTag = 'string'
    keywordTag = 'keyword'
    identifierTag = 'identifier'

    start = 0
    end = searchfrom
    while 1:
        m = search(pytext, end)
        if m is None:
            break    # EXIT LOOP
        start = m.start()
        if start >= searchto:
            break    # EXIT LOOP
        match = m.group(0)
        end = start + len(match)
        c = match[0]
        if c not in "#'\"":
            # Must have matched a keyword.
            if start <> searchfrom:
                # there's still a redundant char before and after it, strip!
                match = match[1:-1]
                start = start + 1
            else:
                # this is the first keyword in the text.
                # Only a space at the end.
                match = match[:-1]
            end = end - 1
            tags_append((keywordTag, start, end, None))
            # If this was a defining keyword, look ahead to the
            # following identifier.
            if match in ["def", "class"]:
                m = idSearch(pytext, end)
                if m is not None:
                    start = m.start()
                    if start == end:
                        match = m.group(0)
                        end = start + len(match)
```

```
                        tags_append((identifierTag, start, end, None))■
            elif c == "#":■
                tags_append((commentTag, start, end, None))■
            else:■
                tags_append((stringTag, start, end, None))■
    return tags■
■
■
def test(path):■
    f = open(path)■
    text = f.read()■
    f.close()■
    tags = fontify(text)■
    for tag, start, end, sublist in tags:■
        print tag, `text[start:end]`■
```