

# Re: Regular Expression – old regex module vs. re module

---

*Source:* <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2006-07/msg00005.html>

---

- *From:* "Steve" <[stever@xxxxxxxxxxx](mailto:stever@xxxxxxxxxxx)>
  - *Date:* 30 Jun 2006 15:49:07 -0700
- 

Hi All!

Thanks for your suggestions and comments! I was able to use some of your code and suggestions and have come up with this new version of Report.py.

Here's the updated code :

```
-----  
#!/usr/bin/env python  
"""Provides two classes to create formatted reports.  
  
The ReportTemplate class reads a template file or string containing a  
fixed format with field tokens and substitutes member values from an  
arbitrary python object.  
  
The ColumnReportTemplate class takes a string argument to define a  
header and line format for multiple calls with sequence data.  
  
6/30/2006  
Steve Reiss (reiss@xxxxxxxxxxx) – Converted to re module methods  
  
"""  
  
__author__ = "Robin Friedrich Robin.Friedrich@xxxxxxx"  
__version__ = "1.0.0"  
  
import string  
import sys  
import re  
  
from types import StringType, ListType, TupleType, InstanceType,  
FileType  
  
#these regex pattern objects are used in the _make_printf function
```

## Re: Regular Expression – old regex module vs. re module

```
exponentPattern = re.compile("\(^|[^\#])|#+\.\#+\*|\*|\*')
floatPattern = re.compile("\(^|[^\#])|#+\.\#+')
integerPattern = re.compile("\(^|[^\#])|##+'")
leftJustifiedStringPattern = re.compile("\(^|[^\<])|<<+')
rightJustifiedStringPattern = re.compile("\(^|[^\>])|>>+')
#####
# _make_printf #
#####
```

```
def _make_printf(s):
    """Convert perl style format symbology to printf tokens.
```

Take a string and substitute computed printf tokens for perl style format symbology.

For example:

```
###.## yields %6.2f
##### yields %8d
<<<<<< yields %-5s
"""
# print("Original String = %s\n\n") % (s)
```

```
while 1: # process all sci notation fields
    if exponentPattern.search(s) < 0: break
    i1 , i2 = exponentPattern.search(s).span()
    width_total = i2 - i1
    field = s[i1:i2-4]
    width_mantissa = len( field[string.find(field,')+1:] )
    f = '%'+`width_total`+'.'+'`width_mantissa`+'e'
    s = exponentPattern.sub(f, s, 1)
```

```
while 1: # process all floating pt fields
    if floatPattern.search(s) < 0: break
    i1 , i2 = floatPattern.search(s).span()
    width_total = i2 - i1
    field = s[i1:i2]
    width_mantissa = len( field[string.find(field,')+1:] )
    f = '%'+`width_total`+'.'+'`width_mantissa`+'f'
    s = floatPattern.sub(f, s, 1)
```

```
while 1: # process all integer fields
    if integerPattern.search(s) < 0: break
    i1 , i2 = integerPattern.search(s).span()
    width_total = i2 - i1
    f = '%'+`width_total`+'d'
    s = integerPattern.sub(f, s, 1)
```

## Re: Regular Expression – old regex module vs. re module

```
while 1: # process all left justified string
fields
if leftJustifiedStringPattern.search(s) < 0: break
i1 , i2 = leftJustifiedStringPattern.search(s).span()
width_total = i2 - i1
f = '%-' + `width_total` + 's'
s = leftJustifiedStringPattern.sub(f, s, 1)

while 1: # process all right justified
string fields
if rightJustifiedStringPattern.search(s) < 0: break
i1 , i2 = rightJustifiedStringPattern.search(s).span()
width_total = i2 - i1
f = '%'+ `width_total` + 's'
s = rightJustifiedStringPattern.sub(f, s, 1)

s = re.sub("\\\\", '\\', s)
# print
# print("printf format = %s") % (s)
return s
```

```
#####
# ReportTemplate #
#####
```

```
class ReportTemplate:
"""Provide a print formatting object.
```

Defines an object which holds a formatted output template and can print values substituted from a data object. The data members from another Python object are used to substitute values into the template. This template object is initialized from a template file or string which employs the formatting technique below. The intent is to provide a specification template which preserves spacing so that fields can be lined up easily.

Special symbols are used to identify fields into which values are substituted.

These symbols are:

```
##### for right justified integer

### for fixed point values rounded mantissa

##### for scientific notation (four asterisks
required)

<<<<< for left justified string
```



## Re: Regular Expression – old regex module vs. re module

```
for i in range(self.nrows):
    splits = string.split(lines[i], self.delimiter)
    body = splits[0] # I don't use tuple unpacking here because
    # I don't know if there was indeed a @@ on the line
```

```
if len(splits) > 1 :
    vars = splits[1]
else:
    vars = "
```

```
#if body[-1] == '\n':
#self.body[i] = body[:-1]
#else:
self.body[i] = body
varstrlist = string.split(vars, ',')
#print i, varstrlist
```

```
for item in varstrlist:
    self.vars[i].append(string.strip(item))
```

```
#print self.vars[i]
if len(self.vars[i]) > 0:
    self.body[i] = _make_printf( self.body[i] )
else:
    print 'Template formatting error, line', i+1
```

```
def __repr__(self):
    return string.join(self.body, "\n")
```

```
def __call__(self, *dataobjs):
    return self._format(dataobjs[0])
```

```
def _format( self, dataobj ):
    """Return the values of the given data object substituted into
    the template format stored in this object.
    """
```

```
# value[] is a list of lists of values from the dataobj
# body[] is the list of strings with %tokens to print
# if value[i] == None just print the string without the %
argument
s = ""
value = []
```

```
for i in range(self.nrows):
    value.append([])
```

```
for i in range(self.nrows):
    for vname in self.vars[i]:
        try:
```

## Re: Regular Expression – old regex module vs. re module

```
if string.find(vname, '[') < 0:
# this is the nominal case and a simple get
will be faster
value[i].append(getattr(dataobj, vname))
else:
# I use eval so that I can support sequence
values
# although it's slow.
value[i].append(eval('dataobj.'+vname))
except AttributeError, SyntaxError:
value[i].append("")

if value[i][0] != "":
try:
temp_vals = []
for item in value[i]:
# items on the list of values for this line
# can be either literals or lists
if type(item) == ListType:
# take each element of the list and tack it
# onto the printing list
for element in item:
temp_vals.append(element)
else:
temp_vals.append(item)
# self.body[i] is the current output line with %
tokens
# temp_vals contains the values to be inserted into
them.
s = s + (self.body[i] % tuple(temp_vals)) + '\n'
except TypeError:
print 'Error on this line. The data value(s) could
not be formatted as numbers.'
print 'Check that you are not placing a string
value into a number field.'
else:
s = s + self.body[i] + '\n'
return s

def writefile(self, file, dataobj):
"""takes either a pathname or an open file object and a data
object.
Instantiates the template with values from the data object
sending output to the open file.
"""
if type(file) == StringType:
fileobj = open(file,'w')
elif type(file) == FileType:
fileobj = file
else:
raise TypeError, '1st argument must be a pathname or an
```



## Re: Regular Expression – old regex module vs. re module

An optional second argument is an output file handle to send written output to (default is stdout). Keyword arguments may be used to tailor the instance. At this time the 'page\_length' parameter is the only useful one.

Instances of this class are then used to print out any number of records with the write method. The write method argument must be a sequence of elements matching the number and data type implied by the field specification tokens.

At the end of a page, a formfeed is output as well as new copy of the header text.

```
"""
```

```
page_length = 50
lineno = 1
pageno = 1
first_write = 1
```

```
def __init__(self, format = "", output = sys.stdout, **kw):
# print("Original format = ", format)
self.output = output
```

```
self.header_separator = re.compile('\n[-_\\s\\t]+\\n')
self.header_token = re.compile('&([^\n\\t]+)')
```

```
for item, value in kw.items():
setattr(self, item, value)
```

```
try: #
use try block in case there is NOT a header at all
result = self.header_separator.search(format).start() #
NEW separation of header and body from format

# print("result = ", result)
HeaderLine = self.header_separator.search(format).group() #
get the header lines that were matched
```

```
if result > -1: # separate
the header text from the format
```

```
# print("split = ", self.header_separator.split(format) )
HeaderPieces = self.header_separator.split(format)
# print("HeaderPiece[0] = ", HeaderPieces[0])
# print("HeaderPiece[1] = ", HeaderPieces[1])
```

```
self.header = HeaderPieces[0] + HeaderLine # header text
```

## Re: Regular Expression – old regex module vs. re module

```
PLUS the matched HeaderLine
self.body = _make_printf(HeaderPieces[1]) # convert the
format chars to printf expressions

except :
self.header = " # fail block of
TRY – no headings found – set to blank
self.body = _make_printf(format) # need to
process the format

# print("header = ", self.header)
# print("body = ", self.body)

self.header = self.prep_header(self.header) # parse the
special chars (&Page &M/D/Y &h:m:s) in header
self.header_len = len(string.split(self.header,'\n'))
self.max_body_len = self.page_length – self.header_len

def prep_header(self, header):
"""Substitute the header tokens with a named string printf
token. """
start = 0
new_header = ""
self.header_values = { }

# print("original header = %s") % (header)
HeaderPieces = self.header_token.split(header) # split
up the header w/ the regular expression

HeadCount = 0

for CurrentHeadPiece in HeaderPieces :

if HeadCount % 2 == 1: # matching
tokens to the pattern will be in the ODD indexes of Heads[]
# print("Heads %s = %s") % (HeadCount,CurrentHeadPiece)
new_header = new_header + '(' + CurrentHeadPiece +')s'
self.header_values[CurrentHeadPiece] = 1
else:
new_header = new_header + CurrentHeadPiece

HeadCount = HeadCount + 1

# print("new header = %s") % (new_header)

return new_header
```

## Re: Regular Expression – old regex module vs. re module

```
def write(self, seq):
    """Write the given sequence as a record in field format.
    Length of sequence must match the number and data type
    of the field tokens.
    """
    seq = tuple(seq)

    if self.lineno > self.max_body_len or self.first_write:
        self.new_page()
        self.first_write = 0

    self.output.write( self.body % seq + '\n' )
    self.lineno = self.lineno + 1

def new_page(self):
    """Issue formfeed, substitute current values for header
    variables, then print header text.
    """
    for key in self.header_values.keys():
        if key == 'P':
            self.header_values[key] = self.pageno
        else:
            self.header_values[key] = now(key)

    header = self.header % self.header_values
    self.output.write('\f'+ header +'\n')
    self.lineno = 1
    self.pageno = self.pageno + 1

def isColumnReportTemplate(obj):
    """Return 1 if obj is an instance of class ColumnReportTemplate.
    """
    if type(obj) == InstanceType and \
        string.find( obj.__class__ , ' ColumnReportTemplate ' ) > -1:
        return 1
    else:
        return 0

#####
# now – return date and/or time value #
#####

def now(code='M/D/Y'):
    """Function returning a formatted string representing the current
    date and/or time. Input arg is a string using code letters to
    represent date/time components.
```

Code Letter Expands to  
D Day of month  
M Month (two digit)  
Y Year (two digit)



## Re: Regular Expression – old regex module vs. re module

```
self.date = now()
# self.time = "18:22:00"
self.time = now('h:m:s') #datetime.time()

self.file = ['TX2667-AE0.dat', 'TX2667-DL0.dat']
self.coeff = -3.4655102872e-05
self.deviation = 0.4018
self.runno = 56 + InValue
self.brkdwn = 43.11
self.invalue = InValue

Report = ReportTemplate(template_string)

for i in range(2):
    D = Data(i)
    print Report(D)

#####
# test_Rt_file – Test Report Template from file #
#####

def test_RT_file():

    template_string = 'ReportFormat1.txt' # filename of report format

    class Data:

        def __init__(self, InValue):
            self.date = now()
            self.time = now('h:m:s') #datetime.time()
            self.file = ['TX2667-AE0.dat', 'TX2667-DL0.dat']
            self.coeff = -3.4655102872e-05
            self.deviation = 0.4018
            self.runno = 56 + InValue
            self.brkdwn = 43.11
            self.invalue = InValue

        Report = ReportTemplate(template_string)

        for i in range(2):
            D = Data(i)
            print Report(D)

#####
# test_CRT – Test Column Report Template #
#####

def test_CRT():

    print
```





## Re: Regular Expression – old regex module vs. re module

```
def Main():

print "\n\nTesting this module.\n\n"

TheHeading = ""
simple heading \#
r–just int fixed point sci–notation left–just string
right–just string
##### #.### #.###**** <<<<< >>>>>"

print
print " Make printf Test : "
print _make_printf(TheHeading)
print
print

test_RT()
test_CRT()

print
test_RT_file()
print

test_CRT2()
test_CRT3()
test_CRT4()

print
print "Current Date & time = ", now('M–D–Y h:m:s')

if __name__ == "__main__":
Main()

.
```