27th February 2012

Regular expressions in Python - the re module

If you have used regular expressions in other languages like Perl, Python regular expressions may seem cumbersome. This article briefly goes over how the Python `re` module is laid out. Note that regular expressions is a vast subject. The article does not deal with how to write regular expressions. If you are not aware of how to write regular expressions http://www.regular-expressions.info/ [http://www.regular-expressions.info/] is an excellent place to start.

`re.compile()` and the `Pattern` Objects

There are two main actors in the `re` module - the `Pattern` object (also called as `RegExObject` in Python documentation) and the `Match` object.

All the usual services that you expect out of regular expressions, such as scanning a text and finding all matches, checking whether there is a match etc., are available as *methods of the* `Pattern` *object*. So, for example, if you want to find all portions of a string that match a particular RegEx, you should first *compile* a `Pattern` object and then invoke its `findall()` method. Here is a trivial example.

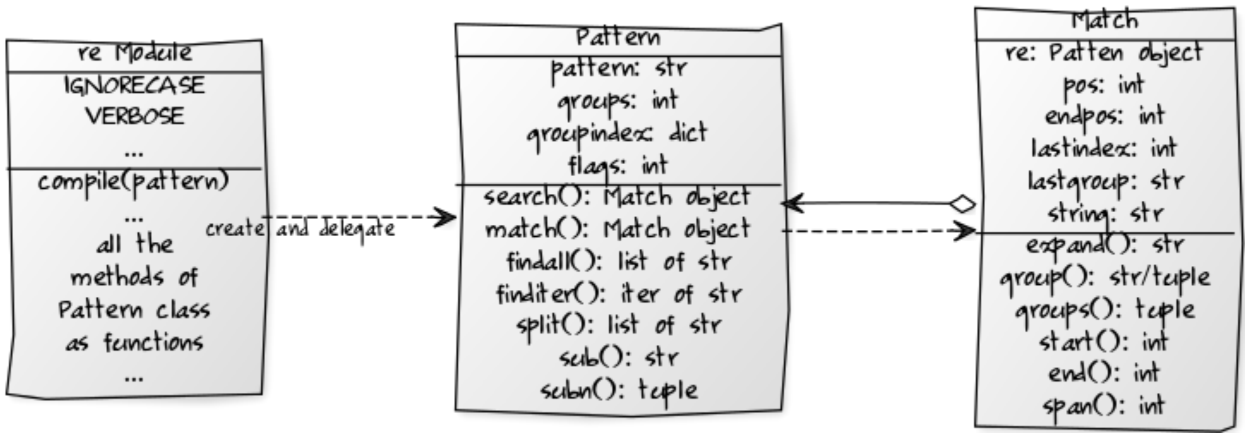
```
>>> patt = re.compile("\w+")
>>> words = patt.findall("Hello World")
>>> words
['Hello', 'World']
```

`re` module *convenience* functions

The `re` module also provides *all* the methods of the `Pattern` object as *convenience functions* directly at the module level. For example, instead of compiling a `Pattern` object and calling the `findall()` method on it, you can directly use the `findall()` *function* in the `re` module. So the example above can also be written as.

```
>>> words = re.findall(r"\w+", "Hello World")
>>> words
['Hello', 'World']
```

Behind the scenes, `re` module just creates a `Pattern` object and calls its `findall()` method.



[http://yuml.me/56d22369]

`match()`, `search()` and the `Match` object

The `match()` and `search()` methods of the `Pattern` object return either `Match` objects if there is a match or `None` if there is no match. `match()` and `search()` are both are very similar except that `match()` looks for a match only at the *beginning* of the string where as `search()` looks for a match anywhere in the string. If there are multiple sub-strings that match the given RegEx pattern, the *first* location that matches is returned as the `Match` object. `Match` objects always have a boolean value of `True`. So in simple cases, if you just want to check if a RegEx pattern matches anywhere in a text, you can use it simply like this.

```
>>> if re.search(r"\w+@\w+\.\w+", "My name is ABC"):
...     print "The text has email address"
... else:
...     print "The text has no email address"
...
The text has no email address

>>> if re.search(r"\w+@\w+\.\w+", "My email is abc@efg.com"):
...     print "The text has email address"
... else:
...     print "The text has no email address"
...
The text has email address
```

`Match` objects have other properties and methods that give a lot more information about the match, such as all the groups that were captured by the match, start and end positions of the string that was matched etc.

Other methods of the `Pattern` object

Pattern object provides other methods `findall()`, `finditer()`, `split()`, `sub()` and `subn()` (which are also available as convenience functions in the `re` module). These methods are simple to use and the documentation is very clear about how they work. They just return a list (or iterator) of the matches or the new string with matched replaced with the alternate text supplied. In most cases, for simple lookups, substituions etc., these methods will do in a pinch.

An example

Here is a simple example that looks for email addresses (and captures the username and domain name). All the the methods and properties of the pattern object, as well as the Match object are printed out. Note: The RegEx pattern to look for email addresses is very simplified and is meant for demonstration only.

```
import re

patt = re.compile(
   r"""
    # The username (capture it)
    (?P<username>\w+)
    # followed by @
    @
    # followed by the domain (also capture it)
    (?P<domain>\w+\.\w+)""",
    re.VERBOSE | re.IGNORECASE)

txt = "My emails are derp@dm1.net and herp@dm2.com."

# print the attributes of the Pattern object
for attr in ['pattern', 'groups', 'groupindex']:
    print 'patt.{0:<12}: {1}'.format(attr, getattr(patt, attr))
print "patt.{0:<12}: {1}".format('flags', bin(patt.flags))
print "patt.search(txt) :\n\t%s" %patt.search(txt)
print "patt.findall(txt):\n\t%s" %patt.findall(txt)
print "patt.split(txt)  :\n\t%s" %patt.split(txt)
print "patt.sub('abc@efg.com', txt):\n\t%s" %patt.sub('abc@efg.com', txt)
print

mtch = patt.search(txt)
# print the attributes and the methods of the Match object
for attr in ['re', 'pos', 'endpos', 'lastindex', 'lastgroup', 'string']:
    print 'mtch.{0:<12}: {1}'.format(attr, getattr(mtch, attr))
for method in ['group', 'groups', 'groupdict', 'start', 'end', 'span']:
    print 'mtch.{0:<12}: {1}'.format(method+'()', getattr(mtch, method)())
```

Here is the output. As you can see, the match object only has details about the first match in the text (`derp@dm1.net`).

```
patt.pattern     :
    # The username (capture it)
    (?P<username>\w+)
    # followed by @
    @
    # followed by the domain (also capture it)
    (?P<domain>\w+\.\w+)
patt.groups      : 2
patt.groupindex  : {'username': 1, 'domain': 2}
patt.flags       : 0b1000010
patt.search(txt) :
 <_sre.SRE_Match object at 0xb788dc80>
patt.findall(txt):
 [('derp', 'dm1.net'), ('herp', 'dm2.com')]
patt.split(txt)  :
 ['My emails are ', 'derp', 'dm1.net', ' and ', 'herp', 'dm2.com', '.']
patt.sub('abc@efg.com', txt):
 My emails are abc@efg.com and abc@efg.com.

mtch.re          : <_sre.SRE_Pattern object at 0xb78b6110>
mtch.pos         : 0
mtch.endpos      : 44
mtch.lastindex   : 2
mtch.lastgroup   : domain
mtch.string      : My emails are derp@dm1.net and herp@dm2.com.
mtch.group()     : derp@dm1.net
mtch.groups()    : ('derp', 'dm1.net')
mtch.groupdict() : {'username': 'derp', 'domain': 'dm1.net'}
mtch.start()     : 14
mtch.end()       : 26
mtch.span()      : (14, 26)
```

Also see

- Doug Hellmann's PyMOTW chapter on re module [http://www.doughellmann.com/PyMOTW/re/]
- The Python Standard Library re module documentation [http://docs.python.org/library/re.html]
- Regular Expression HOWTO [http://docs.python.org/howto/regex.html]

Posted 27th February 2012 by Praveen Gollakota

Labels: compile, match, module, pattern, python, re, regex, regular expression

2 View comments

Shaq February 29, 2012 at 3:19 PM

I saw your name pop up in my feed reader and said, "hey! i know this guy". Thanks for posting this!

Reply

Praveen Gollakota February 29, 2012 at 9:00 PM

Thank you for reading it. Hope it was useful. :)

Reply

Enter your comment...

Comment as: Select profile...

Publish **Preview**