Verbose Regular Expressions

So far you've just been dealing with what I'll call "compact" regular expressions. As you've seen, they are difficult to read, and even if you figure out what one does, that's no guarantee that you'll be able to understand it six months later. What you really need is inline documentation.

Python allows you to do this with something called *verbose regular expressions*. verbose regular expression is different from a compact regular expression in two ways:

- Whitespace is ignored. Spaces, tabs, and carriage returns are not matched as spaces, tabs, and carriage returns. They're not matched at all. (If you want to match a space in a verbose regular expression, you'll need to escape it by putting a backslash in front of it.)
- Comments are ignored. A comment in a verbose regular expression is just like a comment in Python code: it starts with a # character and goes until the end of the line. In this case it's a comment within a multi-line string instead of within your source code, but it works the same way.

This will be more clear with an example. Let's revisit the compact regular expression you've been working with, and make it a verbose regular expression. This example shows how.

```
import re
pattern = '''
    ^                   # beginning of string
    M{0,3}              # thousands - 0 to 3 Ms
    (CM|CD|D?C{0,3})    # hundreds - 900 (CM), 400 (CD), 0-300 (0 to 3 Cs),
                        #            or 500-800 (D, followed by 0 to 3 Cs)
    (XC|XL|L?X{0,3})    # tens - 90 (XC), 40 (XL), 0-30 (0 to 3 Xs),
                        #        or 50-80 (L, followed by 0 to 3 Xs)
    (IX|IV|V?I{0,3})    # ones - 9 (IX), 4 (IV), 0-3 (0 to 3 Is),
                        #        or 5-8 (V, followed by 0 to 3 Is)
    $                   # end of string
    '''

print (re.search(pattern, 'M', re.VERBOSE))                 #①
```

```
#<_sre.SRE_Match object at 0x008EEB48>

print (re.search(pattern, 'MCMLXXXIX', re.VERBOSE))         #②
#<_sre.SRE_Match object at 0x008EEB48>

print (re.search(pattern, 'MMMDCCCLXXXVIII', re.VERBOSE))   #③
#<_sre.SRE_Match object at 0x008EEB48>

print (re.search(pattern, 'M') )                            #④
```

① The most important thing to remember when using verbose regular expressions is that you need to pass an extra argument when working with them: `re.VERBOSE` is a constant defined in the re module that signals that the pattern should be treated as a verbose regular expression. As you can see, this pattern has quite a bit of whitespace (all of which is ignored), and several comments (all of which are ignored). Once you ignore the whitespace and the comments, this is exactly the same regular expression as you saw in the previous section, but it's a lot more readable.

② This matches the start of the string, then one of a possible three `M`, then `CM`, then `L` and three of a possible three `X`, then `IX`, then the end of the string.

③ This matches the start of the string, then three of a possible three `M`, then `D` and three of a possible three `C`, then `L` and three of a possible three `X`, then `V` and three of a possible three `I`, then the end of the string.

④ This does not match. Why? Because it doesn't have the `re.VERBOSE` flag, so the `re.search` function is treating the pattern as a compact regular expression, with significant whitespace and literal hash marks. Python can't auto-detect whether a regular expression is verbose or not. Python assumes every regular expression is compact unless you explicitly state that it is verbose.