Using The {n,m} Syntax

In the previous section, you were dealing with a pattern where the same character could be repeated up to three times. There is another way to express this in regular expressions, which some people find more readable. First look at the method we already used in the previous example.

{1,4} matches between 1 and 4 occurrences of a pattern.

```
import re
pattern = '^M?M?M?$'
print (re.search(pattern, 'M'))     #①
#<_sre.SRE_Match object at 0x008EE090>

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

print (re.search(pattern, 'MMM'))   #③
#<_sre.SRE_Match object at 0x008EE090>

print (re.search(pattern, 'MMMM'))  #④
#None
```

① This matches the start of the string, and then the first optional `M`, but not the second and third `M` (but that's okay because they're optional), and then the end of the string.

② This matches the start of the string, and then the first and second optional `M`, but not the third `M` (but that's okay because it's optional), and then the end of the string.

of the string.

③ This matches the start of the string, and then all three optional `M`, and then the end of the string.

④ This matches the start of the string, and then all three optional `M`, but then does not match the end of the string (because there is still one unmatched `M`), so the pattern does not match and returns `None`.

```
import re
pattern = '^M{0,3}$'                #①
print (re.search(pattern, 'M'))     #②
#<_sre.SRE_Match object at 0x008EEB48>

print (re.search(pattern, 'MM'))    #③
#<_sre.SRE_Match object at 0x008EE090>

print (re.search(pattern, 'MMM'))   #④
#<_sre.SRE_Match object at 0x008EEDA8>

print (re.search(pattern, 'MMMM'))  #⑤
#None
```

① This pattern says: “Match the start of the string, then anywhere from zero to three M characters, then the end of the string.” The 0 and 3 can be any numbers; if you want to match at least one but no more than three `M` characters, you could say `M{1,3}`.

② This matches the start of the string, then one `M` out of a possible three, then the end of the string.

③ This matches the start of the string, then two `M` out of a possible three, then the end of the string.

④ This matches the start of the string, then three `M` out of a possible three, then the end of the string.

⑤ This matches the start of the string, then three `M` out of a possible three, but then *does not* match the end of the string. The regular expression allows for up to only three `M` characters before the end of the string, but you have four, so the pattern does not match and returns `None`.

Checking For Tens And Ones

Now let’s expand the Roman numeral regular expression to cover the tens and ones place. This example shows the check for tens.

```
import re
pattern = '^M?M?M?(CM|CD|D?C?C?C?)(XC|XL|L?X?X?X?)$'
print (re.search(pattern, 'MCMXL'))     #①
#<_sre.SRE_Match object at 0x008EEB48>

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

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

print (re.search(pattern, 'MCMLXXX'))   #④
#<_sre.SRE_Match object at 0x008EEB48>

print (re.search(pattern, 'MCMLXXXX'))  #⑤
#None
```

① This matches the start of the string, then the first optional `M`, then `CM`, then `XL`, then the end of the string. Remember, the (`A|B|C`) syntax means “match exactly one of `A`, `B`, or `C`”. You match `XL`, so you ignore the `XC` and `L?X?X?X?` choices, and then move on to the end of the string. `MCMXL` is the Roman numeral representation of `1940`.

② This matches the start of the string, then the first optional `M`, then `CM`, then `L?X?X?X?`. Of the `L?X?X?X?`, it matches the `L` and skips all three optional `X` characters. Then you move to the end of the string. `MCML` is the Roman numeral representation of `1950`.

③ This matches the start of the string, then the first optional `M`, then `CM`, then the optional `L` and the first optional `X`, skips the second and third optional `X`, then the end of the string. `MCMLX` is the Roman numeral representation of `1960`.

④ This matches the start of the string, then the first optional `M`, then `CM`, then the optional `L` and all three optional `X` characters, then the end of the string. `MCMLXXX` is the Roman numeral representation of `1980`.

⑤ This matches the start of the string, then the first optional `M`, then `CM`, then the optional `L` and all three optional `X` characters, then fails to match the end of the string because there is still one more `X` unaccounted for. So the entire pattern fails to match, and returns None. `MCMLXXXX` is not a valid Roman numeral.

(A|B) matches either pattern A or pattern B, but not both.

The expression for the ones place follows the same pattern. I'll spare you the details and show you the end result.

```
pattern = '^M?M?M?(CM|CD|D?C?C?C?)(XC|XL|L?X?X?X?)(IX|IV|V?I?I?I?)$'
```

So what does that look like using this alternate `{n,m}` syntax? This example shows the new syntax.

```
import re
pattern = '^M{0,3}(CM|CD|D?C{0,3})(XC|XL|L?X{0,3})(IX|IV|V?I{0,3})$'
print (re.search(pattern, 'MDLV') )             #①
#<_sre.SRE_Match object at 0x008EEB48>

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

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

print (re.search(pattern, 'I'))                 #④
#<_sre.SRE_Match object at 0x008EEB48>
```

① This matches the start of the string, then one of a possible three `M` characters, then `D?C{0,3}`. Of that, it matches the optional `D` and zero of three possible `C` characters. Moving on, it matches `L?X{0,3}` by matching the optional `L` and zero of three possible `X` characters. Then it matches `V?I{0,3}` by matching the optional `V` and zero of three possible `I` characters, and finally the end of the string. `MDLV` is the Roman numeral representation of `1555`.

② This matches the start of the string, then two of a possible three `M` characters, then the `D?C{0,3}` with a `D` and one of three possible `C` characters; then `L?X{0,3}` with an `L` and one of three possible `X` characters;

then `V?I{0,3}` with a `V` and one of three possible `I` characters; then the end of the string. `MMDCLXVI` is the Roman numeral representation of `2666`.

③ This matches the start of the string, then three out of three `M` characters, then `D?C{0,3}` with a `D` and three out of three `C` characters; then `L?X{0,3}` with an `L` and three out of three `X` characters; then `V?I{0,3}` with a `V` and three out of three `I` characters; then the end of the string. `MMMDCCCLXXXVIII` is the Roman numeral representation of `3888`, and it's the longest Roman numeral you can write without extended syntax.

④ Watch closely. (I feel like a magician. "Watch closely, kids, I'm going to pull a rabbit out of my hat.") This matches the start of the string, then zero out of three `M`, then matches `D?C{0,3}` by skipping the optional `D` and matching zero out of three `C`, then matches `L?X{0,3}` by skipping the optional `L` and matching zero out of three `X`, then matches `V?I{0,3}` by skipping the optional `V` and matching one out of three `I`. Then the end of the string. Whoa.

If you followed all that and understood it on the first try, you're doing better than I did. Now imagine trying to understand someone else's regular expressions, in the middle of a critical function of a large program. Or even imagine coming back to your own regular expressions a few months later. I've done it, and it's not a pretty sight.

Now let's explore an alternate syntax that can help keep your expressions maintainable.