AndyPatterns

Python list comprehensions vs. the way Ruby does things

Intent: Loop through a list of numbers and pick out the ones greater than 5, and multiply them by 100. Return them as a list.

The Python way...

```python
#!/usr/bin/env python

# Long explicit way:
result = []
for n in range(1,8):
    if n > 5:
        result.append(n*100)
print result
# result [600, 700]

# Concise way:
print [n*100 for n in [1,2,3,4,5,6,7] if n > 5]
# result [600, 700]
```

And the ruby way...

```ruby
#!/usr/bin/env ruby

# Long explicit way:
result = []
(1...8).each do |n|
    if n > 5
        result << n*100
    end
end
p result
# result [600, 700]

# Concise way:
p [1,2,3,4,5,6,7].select{|n| n > 5}.map{|n| n*100}
# result [600, 700]
```

Thoughts:

It seems to me that the puzzle (such that it is) consists of two distinct stages.

The first stage is to select items from the list and the second stage is to apply a transformation onto each selected element.

> I used to think that the python syntax was superior, but once you get around the idea of the two stages, the ruby syntax makes perfect sense too.

Also, I guess the “select + map idiom” is so common that the special python syntax wins in that it isn’t telling you how anything is being done – the “list comprehension” just reads better.

The ruby version is a little more cryptic to the newcomer I guess and is telling you how everything is being done.

The ruby version on the other hand is more general, and would allow other varieties of select/transform/manipulate to get plugged in, and you can have as many as you like e.g.

```
p (1..10).select{|n| n > 5}.map{|n| n+1}.delete_if{|n| n % 2 == 0}
```