Using `returning`, you can write this as:

```
def foo(params)
   returning [] do ¦a¦
      a << 1 if params.include?(:one)
      a << 2 if params.include?(:two)
   end
end
```

I can’t recommend using the second form. The first form is faster, doesn’t create as many interpreter internal data structures as the block-based version, and has the same number of lines.[8]

Using any?

Some people prefer to use `any?` over `empty?` to test for empty strings, arrays, or hashes, apparently because they like the word any better. I can’t recommend that and the following benchmark shows why:

```
n = 1000000
a = [] b = [1, 2, 3]
Benchmark.bm(15) do ¦x¦
   if RUBY_VERSION < '1.9'
      x.report('"".any?'){ n.times{ "".any? }}
      x.report('"".empty?'){ n.times{ "".empty? }}
      x.report('"xyz".any?'){ n.times{ "xyz".any? }}
      x.report('"xyz".empty?'){ n.times{ "xyz".empty? }}
   end
```

[8] *`returning` was inspired by the K combinator known from SKI-calculus. It is defined as λx.λy.x in pure Lambda notation. And who said I couldn’t smuggle some Greek symbols into this book?*

```
    x.report('[].any?'){ n.times{ a.any? }}
    x.report('[].empty?'){ n.times{ a.empty? }}
    x.report('[].blank?'){ n.times{ a.blank }}
    x.report('[1,2,3].any?'){ n.times{ b.any? }}
    x.report('[1,2,3].empty?'){ n.times{ b.empty? }}
    x.report('[1,2,3].blank?'){ n.times{ b.blank? }}
end
```

Results are shown in Table 7.

TABLE 7 **Benchmarking Results Comparing `any?` with `empty?`**

	User	System	Total	Real
““.any?	0.650000	0.000000	0.650000	(0.648618)
““.empty?	0.340000	0.000000	0.340000	(0.343184)
“xyz”.any?	1.440000	0.010000	1.450000	(1.447679)
“xyz”.empty?	0.330000	0.000000	0.330000	(0.338132)
[].any?	0.490000	0.000000	0.490000	(0.485024)
[].empty?	0.230000	0.000000	0.230000	(0.234004)
[].blank?	0.380000	0.000000	0.380000	(0.380161)
[1,2,3].any?	0.920000	0.000000	0.920000	(0.931129)
[1,2,3].empty?	0.230000	0.000000	0.230000	(0.226169)
[1,2,3].blank?	0.390000	0.000000	0.390000	(0.392091)

`string.any?` is so much slower because the character string must be converted into an array first.

Note that Ruby 1.9 no longer supports calling `any?` on strings. This might be a good time to get rid of a bad habit.