Method to Proc

In our Ruby course content on blocks, we learn about "symbol to proc" and how it works. To review briefly, consider this code:

```
comparator = proc { |a, b| b <=> a }
```

and the `Array#sort` method, which expects a block argument to express how the `Array` will be sorted. If we want to use `comparator` to sort the `Array`, we have a problem: it is a proc, not a block. The following code:

```
array.sort(comparator)
```

fails with an ArgumentError. To get around this, we can use the proc to block operator `&` to convert `comparator` to a block:

```
array.sort(&comparator)
```

This now works as expected, and we sort the `Array` in reverse order.

A lone `&` applied to an object causes ruby to try to convert the object to a block. If that object is a proc, the conversion happens automagically, just as shown above. If the object is not a proc, then `&` attempts to call the `#to_proc` method on the object first. Used with symbols, e.g., `&:to_s`, Ruby creates a proc that calls the `#to_s` method on a passed object, and then converts that proc to a block. This is the "symbol to proc" operation (though perhaps it should be called "symbol to block").

Did you know that you can perform a similar trick with methods? You can apply the `&` operator to an object that contains a `Method`; in doing so, Ruby calls `Method#to_proc`.

Using this information, together with the course page linked above, fill in the missing part of the following code so we can convert an array of integers to base 8. Use the comments for help in determining where to make your modifications, and make sure to review the "Approach/Algorithm" section for this exercise; it should prove useful.

```
def convert_to_base_8(n)
  n.method_name.method_name # replace these two method calls
end

# The correct type of argument must be used below
base8_proc = method(argument).to_proc

# We'll need a Proc object to make this code work. Replace `a_proc`
# with the correct object
[8, 10, 12, 14, 16, 33].map(&a_proc)
```

The expected return value of `map` on this number array should be:

```
[10, 12, 14, 16, 20, 41]
```

Hide Solution & Discussion

Approach/Algorithm

To solve this problem you'll need to do a bit of research. Refer to this page on the `Integer` class instance method, `to_s`. You will also have to use one method from the `Object` class. For that, check out this documentation page.

NOTE: This exercise is tricky. If you can't finish it, feel free to follow along with the solution; it may be treated as a learning experience rather than something you should be able to figure out on your own.

Solution

```
def convert_to_base_8(n)
  n.to_s(8).to_i
end

base8_proc = method(:convert_to_base_8).to_proc

[8, 10, 12, 14, 16, 33].map(&base8_proc) # => [10, 12, 14, 16, 20, 41]
```

Discussion

Let's start with our `convert_to_base_8` method. Notice that this method takes a number-like argument, `n`. We also see that `to_s(n)` is using a number-like argument as well. This seems like a good place to start. We'll find that this form of `to_s` converts integers into the String representation of a base-`n` number.

Right now, we use decimals (base 10), so to convert a number `n` to base 8, we call `to_s(8)` on it. If we take 8 as an example, then calling `8.to_s(8)` returns `"10"`. But, from the expected return value, we can see that we want an `Integer`, not a `String`. So, we also need to call `to_i` on the return value of `n.to_s(8)`.

Next, let's handle the missing pieces of this line:

```
base8_proc = method(argument).to_proc
```

Based on the information from the "Approach/Algorithm" section, we know to research `method` from class `Object`. After looking at that documentation, we see that a symbol of an existing method may be passed into method `method(sym)`. If we do that, the functionality of that method gets wrapped in a `Method` object, and we may now do work on that object.

We want to convert our array of numbers to base 8, so it makes sense to make a method object out of the `convert_to_base_8` method. This leaves us with:

```
base8_proc = method(:convert_to_base_8).to_proc
```

The final piece of this exercise asks us fill in this line. `[8,10,12,14,16,33].map(&a_proc)`. We want access to the functionality of method `convert_to_base_8`, and we know that it has been converted to a `Proc` object, so that `Proc` is the natural choice. Remember that using just `&` (and not `&:`) lets us turn a `Proc` object to a block. A block that can now be used with `map`.

There. All done. One last piece of information that may be good to mention is how a method looks when converted to a `Proc`. You can imagine the conversion to look like that:

```
def convert_to_base_8(n)
  n.to_s(8).to_i
end

# ->

Proc.new { |n| n.to_s(8).to_i }
#when we use & to convert our Proc to a block, it expands out to...

# ->
[8, 10, 12, 14, 16, 33].map { |n| n.to_s(8).to_i }
```