RubyGuides

Each vs Map vs Select

These 3 methods (each, map & select) are the most important Ruby methods that you need to know.

Why?

Because a lot of what you are going to be doing is working with data.

And these methods are used to work with collections of data (Array, Hash, Range).

In this guide I want to help you understand the difference between these methods & when you should be using one or the other.

The Each Method

Each is the most basic of the `Enumerable` methods.

In fact, almost every other method that iterates over a collection is based on each.

So it's like a building block.

Let's see an example:

```ruby
numbers = [1,2,3]

numbers.each { |n| puts n }

# 1
# 2
# 3
```

This is your basic loop in Ruby (if you need the index you can use `each_with_index`).

But many times you want to do something more than just print the elements...

That's where the other methods come into play.

The Select Method

Select & Map are specialized versions of each.

With select you can "filter" your array to create a new array with only the matching elements.

Example:

```
numbers = [1,2,3,4,5,6]

numbers.select { |n| n.even? }

# [2,4,6]
```

In this example we filter our original numbers to only the numbers that are even.

We can still do this with the each method, but it will require some extra code.

Example:

```
numbers = [1,2,3,4,5,6]
results = []

numbers.each { |n| results << n if numbers.even? }

p results
#[2,4,6]
```

Notice how we need a few extra elements:

- An empty array to store our results
- The << (called "shovel" or "push" operator), which appends a new value into the results array
- An if statement

All of these things are taken care for us by Ruby when we use `select` instead of each.

That's what I mean with "a more specialized version of each".

The Map Method

The `map` method helps you collect the results of iterating over your array elements, resulting in a new array.

It can be used to apply some operation, like `downcase` or `upcase`, to every element of the array.

Example:

```
names = ["David", "Grant", "Bob"]

names.map { |name| name.downcase }

# ["david", "grant", "bob"]
```

Notice that this doesn't change the original array! It returns a new one. If you want to save the results of `map` you need to use a variable.

Like this:

```
downcase_names = names.map { |name| name.downcase }
```

You can also use `map` for numbers or anything else:

```
numbers = [1,2,3,4,5]

numbers.map { |n| n * 10 }

# [10, 20, 30, 40, 50]
```

Don't just read this. Open `irb` now & practice using `map` :)

The Difference

Ok so what is the difference between these?

Besides each, every `Enumerable` method has it's own specialized function.

Refer to the Ruby documentation to learn what every method does & to see examples.

> (Note: `Enumerable` is a built-in Ruby module that hosts these methods)

You should always use the more specialized methods if possible because they are more efficient & produce better code.

If you can't find the perfect method for your situation remember that can you always fall-back to a more basic method like each (or even a `while` loop) & change it later when you find a better method.

> (Watch this video to learn more about loops.)

Another important thing is knowing what the method returns.

Map & select will return a new array, while each will always return the original array.

Summary

You have learned about some of the most important Ruby methods (each, select & map), how they work & the differences between them.

Important points to remember:

- map & select DON'T change the original array (unless you use `map!` or `select!`, there is no `each!`)
- each is a building block for the other `Enumerable` methods
- `Enumerable` methods are the key for writing idiomatic Ruby code

If you have mastered those already look into the following methods: `inject`, `partition` & `group_by`.

Exercise 1:

Write a Ruby program that reads in a list of names & then prints only the names that have a string size of 4 or above.

Exercise 2:

Just like I did with the `select` method, try to understand what `map` is doing for you & write your own version of the `map` method.