Ruby Map Enumerable Method

Map is a Ruby method that you can use with Arrays, Hashes & Ranges.

The main use for map is to **TRANSFORM** data.

For example:

Given an array of strings, you could go over every string & make every character UPPERCASE.

Or if you have a list of `User` objects…

You could **convert** them into a list of their corresponding email addresses, phone number, or **any other attribute** defined on the `User` class.

Ruby Map Syntax

The syntax for map looks like this:

```
1.array = ["a", "b", "c"]

2.array.map { |string| string.upcase }

3.# ["A", "B", "C"]
```

First, you have an array, but it could also be a hash, or a range.

Then you call `map` with a block.

The block is this thing between brackets `{ ... }`. Inside the block you say HOW you want to transform every element in the array. It's basically a function.

What happens after you call `map`?

Map **returns a new array** with the results.

It won't change the original.

If you want to change the original array you can use **`map!`**.

Ruby Map Examples

Here are some examples that you may find useful.

Doubling numbers:

```
1.array = [1,2,3]

2.array.map { |n| n * 2 }

3.# [2, 4, 6]
```

Convert strings to integers:

```
1.array = ["11", "21", "5"]

2.array.map { |str| str.to_i }

3.# [11, 21, 5]
```

Convert hash values to symbols:

```
1.hash = { bacon: "protein", apple: "fruit" }

2.hash.map { |k,v| [k, v.to_sym] }.to_h

3.# {:bacon=>:protein, :apple=>:fruit}
```

About this hash example:

You'll find that we have two arguments instead of one, that's because a hash element is composed of a key & a value.

Then I'm returning a new array with the transformed key & values.

The last step is to convert this back into a hash.

Ruby Map vs Each

What is the difference between map & each?

Each is like **a more primitive version of map**…

It gives you every element so you can work with it, but it doesn't collect the results.

Each **always returns the original, unchanged object**.

While map does the same thing, but…

It returns a new array with the transformed elements.

Example:

```
1.array.each { |n| n * 2 }

2.# [1, 2, 3]

3.array.map { |n| n * 2 }

4.# [2, 4, 6]
```

Ruby Map vs Collect

Map and Collect are exactly the same method.

They are different names for the same thing!

Which one should you use?

If you read open-source projects you'll find that the most common version is `map`.

Use that.

How to Use Map With an Index

If you need an index with your values you can use the `with_index` method.

Here's an example:

```
1.array = %w(a b c)

2.array.map.with_index { |ch, idx| [ch, idx] }

3.# [["a", 0], ["b", 1], ["c", 2]]
```

Bonus tip:

You can pass a parameter to `with_index` if you don't want to start from index 0.

Ruby Map Shorthand (map &)

You can use a shorthand version for `map` when you're calling a method that **doesn't need any arguments**.

Example:

```
1.["11", "21", "5"].map(&:to_i)
```

Example:

```
1.["orange", "apple", "banana"].map(&:class)
```

This & syntax is not limited to `map`, it can also be used with other enumerable methods.

You've heard of #**`map`**? It's the EXACT same method as collect, just called something different. Some people visualize it in their heads as doing something and collecting the results, other people see it as re-mapping your original object through some sort of transformation. It's more conventional to use #`map` but both work the same way.

Here's a theoretical example more like what you might see when you've got your own website built using Rails, where we may want to send only an array filled with our users' emails out to the webpage:

```
u = User.all
@user_emails = u.map { |user| user.email }
```

You can also use these methods on hashes as well, just remember that now you have two inputs to work with for your block:

```
> my_hash = {"Joe" => "male", "Jim" => "male", "Patty" => "female"}
> my_hash.select{|name, gender| gender == "male" }
=> {"Joe" => "male", "Jim" => "male"}
```

Ruby Map Enumerable Method