# 9 Ruby: Loops

A loop is a programming construct that is used to "step through" a collection of objects. The proper way to express this in programming terms is to say that a loop "iterates over a collection."

As we know, an `Array` is a collection of objects, so we could use a loop to iterate over an `Array`. While iterating over a collection of objects, a loop can be programmed to do things to or with each object in the collection.

## Each

There are different ways to program loop constructs in Ruby, but the most common is `each`. The `each` method is called on a collection, like an `Array`. Try the following using **repl.it**:

```
a = [2, 4, 6, 8]
a.each do |num|
  p "We're on number: #{num}"
end
```

Here's what you did:

- First, we created a variable named `a` and assigned it to an `Array` containing four numbers.
- On the second line of code, we called the `each` method on the `a` array and passed it a `block` of code. A `block` starts with a `do` and ends with an `end`.
- On the second line of code, after `do`, we declared a variable named `num` to be used in the `block`. This variable represents the element over which the `each` method is iterating.

- The third line of code is inside of the `each` block. We can think of this line as being "in the loop". Inside of the loop we are instructing the computer to print an interpolated string with the `num` argument. Remember, the `num` variable represents an element from the `a` array. With each iteration, `num` is assigned to a new element. In this example, `num` is assigned to `2`, then `4`, then `6`, then `8`.
- The fourth line of code closes the `each` block with the keyword `end`.

> A deeper discussion of `blocks`, which we've already encountered by learning to read RSpec, is coming up shortly. For now, make sure you're comfortable with its basic syntax.

## Each in Rails

Each is frequently used in Rails to iterate through "model" objects (instances of classes representing data) and display their attributes on a page. We haven't yet encountered model objects, and we don't yet have a web page, but we can explore usage like this in the below example, which shows how we might loop through `User` objects to display their information using `each`:

```
class User
  attr_accessor :name

  def initialize(name)
    @name = name
  end

  def email
    # You'll get a chance to use `join` and `split` in the exercises
    "#{name.split(' ').join('')}@email.com"
  end
end

users = [User.new('Harry Potter'), User.new('Hermione Granger'), User.new('He Who Sh

users.each do |user|
  p "This user's name is #{user.name}, and their email is #{user.email}."
end
```

This would loop through the `users` collection and output a string (ex: "This user's name is Harry Potter, and their email is HarryPotter@email.com.") for each one. Because `each` is a method, it returns something at the end of this loop: the same collection of `users` on

which it was originally called.

> `each` performs operations on a collection, but returns the original collection, rather than anything related to what is performed within the `do` and `end` of the block. In this sense, we'd say `each` is used for its "side effects", rather than its "return".

## Mixing Classes and Loops

Let's combine loops and classes in practice. We'll repeat the above behavior, but move some of it into a new class. This class will take an array of user objects on initialization and will have a `print_names_and_emails` method, which will loop through `users` and print out a string for each.

```
# User class defined above

class UserIterator
  attr_accessor :user_array

  def initialize(user_array)
    @user_array = user_array
  end

  def print_names_and_emails
    user_array.each do |user|
      p "This user's name is #{user.name}, and their email is #{user.email}"
    end
  end
end

users = [User.new('Harry Potter'), User.new('Hermione Granger'), User.new('He Who Sh

UserIterator.new(users).print_names_and_emails
```

This example would behave identically to our example above it. It would print out a string for each member of the collection, as a "side effect". Then the loop inside the `print_names_and_emails` method would return the `user_array` on which it was called. Because this call of `each` is the last thing that happens in the `print_names_and_emails` method, it is also, by default, what that method returns. So, just like our example without the `UserIterator` class, calling `UserIterator.new(users).print_names_and_emails` returns the original array of `users` with which we initialized the class instance.

Assignment | Discussion | Submission

## Exercises

- [x] ~~The Each Method~~
- [x] ~~Each With Index~~
- [x] ~~Title Case~~

## Solution Walk-Throughs

**Do not watch these videos until after you've attempted your own solutions.** If you struggle to complete these exercises, submit your best effort to your mentor *before watching a solution walk-through*. Submit your answers even if some tests don't pass, as it's important for your mentor to understand your thought process. Discuss difficult exercises with your mentor.

There are many possible solutions to the exercises. A walk-through is meant to provide insight into the thought process of *one possible solution* per exercise. If your solution is different yet passes all the tests, do not change your solution to match what is seen in the video. Instead, discuss your thought process with your mentor.

1. **Each With Index Solution**
2. **The Each Method Solution**
3. **Title Case Solution**

assignment completed

?