Link to lesson

A link to the lesson can be found here

Iterators

- Both **hahes** and **arrays** in ruby are known as *Collections*.
- Collections have predefined methods which allow us to iterate over them.
- These methods are known as **iterators**.

Iterator Syntax

1. Iterator syntax example with keywords **do** and **end**.

```
collection.each do |item|
    code to be executed per iteration
end
```

2. **Shorthand** with curly braces replacing keywords **do** and **end**.

```
collection.each { |variable| code }
```

- It's best to use the first examples syntax for a **multi-line block of code**.
- Shorthand works well for **single line code**.

Three parts to any iterator

Example block of iteration:

```
collection.each do |item|
  code to be executed per iteration
end
```

1. **METHOD**

- **each** is the method in the example above. This will iterate over every element in the **collection**.

```
.each
```

2. **VARIABLE**

- **item** is the variable in the example above and is represented by being place between two | | (referred to as **PIPES**).

```
|item|
```

3. **CODE/BLOCK**

- This is a **block of code** that is executed with every iteration.

```
code to be executed per iteration
```

Block

- A **Block** is a piece of code that is executed between the keywords **do** and **end** OR between {} referred to as *curley braces*.
- Blocks of code get executed for each iteration of the collection.
- In other words, the code that is being executed for every iteration is called a **Block**.

Methods

- The **each** method iterates through every element in the *collection* (Hash or Array) and then executes the code block.
- Each iterator for an **array** takes **ONE** argument.
    - item
- Each iterator for a **hash** takes **TWO** arguments.
    - key and value

Each method with Array

Example 1 - Array with each method

```
numbers = ["one","two","three","four"]

numbers.each do |item|
    p item
end

=> "one"
=> "two"
=> "three"
=> "four"
```

Example 2 - Array with each method

```
cities = ["Sydney", "Melbourne", "Brisbane"]

cities.each do |city|
puts "The city is #{city}"
end
```

- As examplified, the **variable between the pipes** can be named ANYTHING
- BUT it is BEST PRACTICE to name it something related to the **variable** to be iterated on.
- In this manner, example 2 (above) has a **better naming convention** than example 1 (above).

Each method with Hash

Example 1 - Hash with each method

```
    capital_cities = {:nsw => "Sydney", :vic => "Melbourne", :qld =>
"Brisbane"}

    capital_cities.each do |key, value|
        p key, value
    end

    => :nsw
    => "Sydney"
    => :vic
    => "Melbourne"
    => :qld
    => "Brisbane"
```

Example 2 - Hash with each method

```
    capital_cities = {:nsw => "Sydney", :vic => "Melbourne", :qld =>
"Brisbane"}

    capital_cities.each do |state, city|
        puts "the capital city of #{state} is #{city}
    end

    => the capital city of nsw is Sydney
    => the capital city of vic is Melbourne
    => the capital city of qld is Brisbane
```

each_with_index

- **each_with_index** is similar to the **each** method, however it takes TWO arguments from an **array** and three arguments from a **hash**:
    - ARRAY with each_with_index - The **value** of an array and its **index** (see example below).

```
numbers = ["one","two","three","four"]

numbers.each_with_index do |item, index|
        p item
        p index
end
```

    - HASH with each_with_index - The **key**, **value** and **index** for a hash (see example below).

```
capital_cities = { :nsw => "Sydney", :vic "Melbourne", :qld =>
"Brisbane"}

capital_cities_each_with_index do | (key, value), index |
        p key, value, index
end
```

array.each_with_index

Example 1:

```
    cities = ["Sydney", "Melbourne", "Brisbane"]

    cities.each_with_index {|city,index| puts "The city is #{city} at
index position #{index}"}
```

Returns:

```
    => The city is Sydney at index position 0
    => The city is Melbourne at index position 1
    => The city is Brisbane at index position 2
```

hash.each_with_index

- When we are passing the three arguments **key**, **value** and **index**, it is important to place the **key**, **value** inside () *referred to as parentheses*.

  ```
            | (key, value), index |
  ```

Example 1 -hash.each_with_index

```
    capital_cities = { :nsw => "Sydney", :vic => "Melbourne", :qld =>
"Brisbane"}

    capital_cities.each_with_index do |(state, city), index|

    puts "The capital city of #{state} is #{city}"
    puts "Index position is #{index}"

    end
```

Returns:

```
    => The capital city of nsw is Sydney
    => Index position is 0
    => The capital city of vic is Melbourne
    => Index position is 1
    => The capital city of qld is Brisbane
    => Index position is 2
```

- Most of the time, this method to find the index is NOT used with a **hash**.
- Instead, we use this method with an **array**.

Map

- Map is an **iterator method** SIMILAR to **each** with ONE KEY DIFFERENCE.
- It ALSO iterates through EVERY element within an **array** or a **hash**.
- Map creates a BRAND NEW ARRAY from **values** returned by the **block**.

Example 1:

```
numbers = ["one","two","three","four"]

numbers_uppercased = numbers.map do |item|
        item.upcase
end

p numbers
p numbers_uppercased
```

Result:

```
=> ["one", "two", "three", "four"]
=> ["ONE", "TWO", "THREE", "FOUR"]
```

Example 2:

```
names = ["bob","charlie", "Amy"]

capitalized_names = names.map {|name| name.upcase}

p names
p capitalized_names
```

Result:

```
=> ["bob", "charlie", "Amy"]
=> ["BOB", "CHARLIE", "AMY"]
```

Key Difference between Map and Each

- The KEY difference is we will use MAP when want want to **generate a NEW array** from the original array.
- However, if we are just interested in **iterating through an element** and performing a **certain action** and NOT interested in generating a new array, we can use the EACH method.

Filter/Select

- **select** return a NEW array with the TRUTHY results.

Example 1: Normal syntax

```
[1,2,3,4,5].select do |num|
        num.even?
end
```

Example 2: Shorthand syntax

```
[1,2,3,4,5].select {|num| num.even?}
```

Result (for both examples)

```
=> [2,4]
```

- Whenever we are using a **filter** or a **select** method, we NEED a **truth or false condition**.

  ```
  num.even?
  ```

- This will return the even numbers in the array.

  ```
  [2,4]
  ```

OR WE CAN USE ANOTHER TRUTHY CONDITION

```
num.odd?
```

- This will return the odd numbers in the array.

  ```
  [1,3,5]
  ```