# 10 Ruby: Hashes

Arrays and Strings allow their elements to be accessed by integers. The integers specify an index in the Array or String. There is another type of collection in Ruby, called a `Hash`.

Hashes store elements as key / value pairs. A key is a unique identifier, and in this respect acts like an index. Keys don't have to be integers, however. In fact, they can be a number of different objects, such as a `String` or `Symbol`. In a `Hash`, you use a **key** to retrieve a **value**.

`Symbols` are the most common types of hash key. Symbols are essentially lightweight strings, denoted with a `:` to the left of their name. To create a `Symbol` named "book", we would write `:book`.

The example below demonstrates how to use Symbols as keys to retrieve values from a `Hash`. The code below is a dictionary program that returns the definition of a word (type of fruit) that's passed to it.

> Dictionaries are a classic use case for Hashes. In fact, this type of data structure is often referred to as a dictionary in computer science.

```
def get_definition(word)
  fruits = {:apple => "A red, yellow or green fruit.", :banana => "A yellow fruit.",
  p fruits[word]
end

# Call the method, requesting the definition of an apple
get_definition(:apple)
#=> "A red, yellow or green fruit."
```

In the example above we passed `:apple`, a `Symbol`, to the `get_definition` method. The `:apple` argument was a valid key in the `fruits` `Hash`, so it returned its definition. The definition in this case is the value in the `fruits` `Hash` associated with the `:apple` key. We extract the value for a given key like this:

```
fruits[:apple] #=> returns "A red, yellow or green fruit."
```

`fruits` is the `Hash` and `:apple` is the key whose value we want. The syntax is similar to the array indexing syntax:

```
['one', 'two', 'three'][0]
#=> 'one'
```

## Shorthand Hash Syntax

If you use a symbol as a key in `Hash`, you can place the colon on the right and remove the "hash-rocket" (`=>`). Using this shortcut, the `fruits` `Hash` in the `get_definition` method could be rewritten as:

```
def get_definition(word)
  fruits = {apple: "A red, yellow or green fruit.", banana: "A yellow fruit.", water
  p fruits[word]
end
```

> Though either `Hash` syntax will work, the latter (without the hash-rocket) is increasingly well accepted and offers greater readability.

When Hashes become very long, or verbose, you can reformat them to make the code more readable. For example:

```ruby
def get_definition(word)
  fruits = {
    apple: "A red, yellow or green fruit.",
    banana: "A yellow fruit.",
    watermelon: "A large green fruit."
  }
  p fruits[word]
end
```

In an `Array`, order is very important; in a `Hash`, it is not. A `Hash` is stored without regard to order, so keys are all that you need to access a value.

10. Ruby: Hashes

| Assignment | Discussion | Submission |
|---|---|---|

## Exercises

- ~~Setting Attributes~~
- ~~Iterating over a Hash~~
- ~~Hash Methods~~

As an added assignment to reinforce your comfort with hashes, send your mentor a message — 500 words max — explaining in your own words what hashes are, why they're useful, and the syntax used to access their elements.

# 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. **Setting Attributes Solution**
2. **Iterating over a `Hash` Solution**
3. `Hash` **Methods Solution**

assignment completed