Open in new tab

Hashes

Lesson Duration: 60 minutes

Learning Objectives

- Create a hash
- Add items to a hash
- Retrieve items from a hash
- Delete items from a hash
- Understand what a symbol is

What Are Hashes

We have seen that we can store a collection of objects in an array object. Let's say we wanted to store the food we are going to have that day.

Let's create a file called hashes.rb

```
# hashes.rb
meals = [ 'yoghurt', 'roll', 'steak' ]
p meals[0]
# => 'yoghurt'
```

This is fine but we need to remember that breakfast is index 0, lunch is index 1 and dinner is index 2. We can only access items by their index in an array.

In our example, we actually have additional information about the objects in our collection. We know the first one is breakfast, the second is lunch and the third is dinner. We don't actually care which order they are stored in really. Also, if we forgot the position of breakfast we would need to search through the whole set to find it again!

In Ruby, and many other languages, there is another collection we can use: a Hash. It's sometimes called a Dictionary or Associative Array in other languages.

In a hash every item is given a unique key of our choosing and it is this key that is used to retrieve the object not an index. The order is irrelevant so we can't rely on the ordering.

Each key in the hash is unique allowing you to always find the value you stored against a particular item.

It's a little bit like a filing cabinet - we have labels we associate with things we want to store e.g. finance, recipes, reciepts. It doesn't matter what "index" the items are stored at (there's no need to know Finance is the first set of items in the drawer), what matters is the label we filed it under. If we moved all the items around, we could still find "Finance" easily by its label.

Using Hashes

Creating Hashes

We have a couple of options for initialising an empty hash. We can 'new' it up, or we can create an empty hash directly.

```
# hashes.rb
my_first_hash = Hash.new()
my_second_hash = {}
```

Let's create a populated hash!

```
# hashes.rb
meals = { "breakfast" => "yoghurt",  "lunch" => "roll", "dinner" => "steak" }
p meals
```

In this hash, have have keys which are string and values which are strings. The keys could be other types of things, like Integers, and the values could be any object - Arrays, Booleans, anything.

```
# hashes.rb
silly_thing = { 1 => "2", 2 => "3", 4 => false }
p silly_thing
```

Accessing Elements

We can access elements in a similar manner to arrays, using the square brackets - the main difference is, we have to use the key, instead of an index.

```
# hashes.rb
p meals["breakfast"]
```

If we try to access an element for which there is no key the hash will return `nil`.

```
# hashes.rb
p meals["supper"]
```

This is a default value that is returned when accessing keys that do not exist in the hash. We can override this value if we wish by passing our own value as the argument when we create it:

```
# hashes.rb
meals = Hash.new(0)
```

Now it will always return 0 when a key is not found.

Modifying Elements

We can add objects to a hash much like we would assign variable.

```
# hashes.rb
meals["supper"] = "pancakes"
p meals
```

We can also replace objects

```
# hashes.rb
meals["dinner"] = "pasta"
p meals
```

We can remove items using the delete method.

```
# hashes.rb
meals.delete("breakfast")
p meals
```

> [TASK:] Make a hash with a key of a persons name and the value as their pocket money. Try updating and deleting items from it.

Helpful Methods

A hash has lots of helpful methods, including a way to list all the keys:

```
# hashes.rb
p meals.keys
```

Can you guess the type of what has been returned? Yes! An Array.

We can also get the list of values:

```
# hashes.rb
p meals.values
```

Symbols

Symbol is a class provided to us by Ruby when objects are particularly suited as keys for hashes. A Symbol is essentially a special string, but it's a constant value. We can never alter the symbol by adding things to it or do many of the other functions we might expect to use on a string. Symbol comparison is a lot faster than String comparison so they are ideal for hash keys.

We won't get bogged down in this, but it's a very common thing in Ruby and when you start using symbols you won't want to go back to strings for keys.

```
# hashes.rb
p :my_sym
p :hello

p :my_sym + :hello #NOPE
```

```
# hashes.rb
meals = { :breakfast => "yoghurt",  :lunch => "roll"  }
p meals
```

Symbols are so commonly used as keys in hashes that Ruby gives us a special syntax.

```
# hashes.rb
meals = { breakfast: "yoghurt",  lunch: "roll" }
p meals[:breakfast]
```

Nested Hashes

Let's make a quick hash of countries.

```
# hashes.rb

countries = {
    uk: "London",
    germany: "Berlin"
  }

p countries
```

This is fine, but what if we also want to store the population? Would we make a separate hash?

We can actually store a hash inside of a hash! Sounds scary, but it can actually be very useful. Let's add the populations.

```
# hashes.rb

countries = {
        uk: {
                capital: "London",
                population: 1000
        },
        germany: {
                capital: "Berlin",
                population: 5
        }
}

p countries
```

Task

See if you can figure out how to get the population out of Germany.

```
# hashes.rb
countries[:germany][:population]
p countries
```

Task

Make a hash to store the superhero avengers. It should contain keys for Iron Man and Hulk.

Each avenger is represented by another hash, and has a name (Tony Stark and Bruce Banner respectively) and another hash containing their attacks.

Each attack should have an integer value of the attack's power. Iron man can punch (10) and kick (100) and Hulk can smash (1000) and roll (500).

Once you have created the hash retrieve and print out the attack power of Hulks smash move.

Solution

To create the hash:

```
#avengers_hash.rb

avengers = {
  ironman: {
    name: "Tony Stark",
    moves: {
      punch: 10,
      kick: 100
    }
  },
  hulk: {
    name: "Bruce Banner",
    moves: {
      smash: 1000,
      roll: 500
    }
  }
}
```

To retrieve the attack power of the hulk's smash move from the hash:

```
#avengers_hash.rb
# attack power of hulk's smash move
p avengers[:hulk][:moves][:smash]
```

Published with GitHub Pages