Exercises - Session 13

In case you get stuck anywhere, don't be afraid to ask the coaches! They are here to help and will gladly explain everything to you!
Take notes during the exercises. Even if you never look at them again, they will help you memorise things!

Arrays and Hashes - when to use which?

Arrays and hashes are the bread and butter of the Ruby programming language. They both represent a collection of things, but they are meant for different use cases.

When thinking about collections, there are three main use cases one may try to use them for:

- Do something with each element in the collection:
  `['abc', 'def'].map { |element| element.upcase } # => ["ABC", "DEF"]`
- Access a specific element that you know beforehand:
  `['abc', 'def'].first # => 'abc'`
- Access an element, but you don't know which one: You need to search in one form or another through the collection until you found the appropriate element:
  `['abc', 'def'].find { |element| element == 'abc' } # => 'abc'`

These are some of the aspects to check when you need to decide which to use:

- If you have 2 things associated (a key to a value; such as a place and its zip code) you probably want to use a hash.
- If you want to apply a certain action to all the elements, go with an array. The same thing is true if you care about the order of things.
- Search:
  - Your search may return multiple results, you need to go with an array. You cannot associate multiple values to the same key in a hash
  - Your collection to search through is small (thousands), go with an array
  - Your collection is big (between thousands and million) and it is easy to associate the result with the search-term, build your dataset as a hash.
  - Any search through a bigger dataset should be handled completely different (with a database query for example)

The examples below are deliberately awkward to solve with either hash or an array. Still, please try to solve them with a hash and an array. It should be very clear which one is the appropriate solution. Please ask if it's not clear!

1. You have a list of places with their zip code. You need to find the name for a given zip code (e.g. 4000). Do this with an array and with a hash. Which one is easier?

array: `places = [3000, "Bern", 4000, "Basel", 6000, "Luzern"]`
hash: `places = {3000 => "Bern", 4000 => "Basel", 6000 => "Luzern"}`

2. Rather than having a zip-code you have the name of the city (e.g. “Basel”). What changes to the previous example? Do you have to change considerably less with one solution? Which one? Feel free to change anything with the given example that makes your life easier. Even completely reorganizing the content of the array or the hash. The only thing we ask is that you use an array and a hash once.

   array: `places = [3000, "Bern", 4000, "Basel", 6000, "Luzern"]`
   hash: `places = {3000 => "Bern", 4000 => "Basel", 6000 => "Luzern"}`

3. Replace all `"B"` with `"P"`. The data structure should be the same afterwards.

   array: `places = [3000, "Bern", 4000, "Basel", 6000, "Luzern"]`
   hash: `places = {3000 => "Bern", 4000 => "Basel", 6000 => "Luzern"}`

   Hint: Read up on `each_pair` http://docs.ruby-lang.org/en/2.0.0/Hash.html#method-i-each_pair to get an idea how to do this with a hash

Method Extraction

Let’s consider this code:

```
def parse_input(input)
  lines = input.split("\n")

  lines.each do |line|
    parts = line.split(":")
    name = parts[0]
    items = parts[1].split(",")

    @person_item_counter[name] = {}

    items.each do |item|
      stripped_item = item.strip
      number_and_label = stripped_item.split(" ")
      item_number = number_and_label[0].to_i
      item_label = number_and_label[1]
      @item_counter[item_label] += item_number
      @person_item_counter[name][item_label] = item_number
```

```
    end
  end
end

input = [
  "Ferdinand: 3 sweaters, 8 teas, 2 visits",
  "Fabian: 2 sweaters, 4 teas, 3 visits",
  "Sascha: 2 sweaters, 6 teas, 4 visits",
  "Thomas: 3 sweaters, 8 teas, 2 visits"
].join("\n")

@item_counter = {"sweaters" => 0, "teas" => 0, "visits" => 0}
@person_item_counter = {}

parse_input(input)
```

That's quite a long method! Although it's still practical to read through it, it makes sense to extract methods out of this. Here's a few hints and thoughts that might help you guide through the process:

- What does this code actually do? How can you find out?
- Can you explain every line of code? If not, what don't you understand? Ask a coach if you can't figure it out.
- What are lines of code that belong together?
- What method name would be fitting to describe those lines that belong together?
- What should the method ideally take as an argument/arguments?
- What should the method return? Does it need to return anything?