Exercises - Session 9

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!

Recap previous sessions

1. With the following code you would normally iterate over an array:

   ```
   places = ["Paris", "Nairobi", "Tokyo", "Portland"]
   places.each do |a_place|
     puts "Place: #{a_place}"
   end
   ```

   Find a way to achieve the same output without using the "each" method. Your code also has to work without changes with more or less elements in the "places" array.

Methods

1. Write a method "greet" that takes a name as an argument and says "Hello" and then the name, followed by an exclamation mark.

2. Define a method that tells you which person lived longer.

   A person is defined as a hash, like this:

   ```
   beethoven = { "name" => "Ludwig van Beethoven", "year_born" => 1770,
   "year_died" => 1827 }
   kant = { "name" => "Immanuel Kant", "year_born" => 1724, "year_died" => 1804 }
   ```

   Your method takes two people as arguments and returns for the example above either:

   "Ludwig van Beethoven grew older than Immanuel Kant!"

   or

   "Immanuel Kant grew older than Ludwig van Beethoven!"

   Bonus Question: Are these the only two possibilities? If not, how could you handle the other cases?

3. We have the following method:

```
def mystery(number)
  changed = number * 5
  if changed % 3 == 0
    “Bam!”
  else
    “Zonk!”
  end
end
```

What does the method “mystery” return if we call it with…

a) `mystery(3)`

b) `mystery(4)`

Try to first solve this exercise by thinking about the code and using pen and paper. Then implement it in a Ruby program and see whether you were right!

Optional Part

1. Write a method called “translate” that takes a word and returns the translation. If the translation could not be found, it should return: “I’m sorry, I can’t translate that.”

   Tip: Use a Hash to store your translations!

2. OK, this is a tricky one, so don’t be discouraged if you don’t find a solution. And let the coaches help you!
   Methods can also call themselves. This is called **Recursion** and is a concept that can be very useful for some problems.
   The following method is not a recursive method, but an iterative one (because it has a loop). It calculates the factorial for a number x. For example 4! == 4 * 3 * 2 * 1 == 24.

```
def factorial(x)
  f = 1

  (1..x).each do |i|
      f = f * i
  end

  f
end
```

   Here’s a challenge for you: Can you write a recursive method that calculates a factorial?