Concurrent Programming in Erlang

In this exercise we'll look in practice at how the mailbox is handled in Erlang. In particular, we'll look at how messages can be processed in the order that they arrive in, as well as in the order that we choose to use.

Use the comments on this step to discuss your strategies, ask any questions and share your solutions.

Processing messages in mailbox order

The simplest way of dealing with messages from a mailbox is to deal with them in the order in which they are received. We can guarantee this by matching with a variable in a `receive` statement, like this:

```
receive
    X -> … how to process the message X
…
end
```

(To be pedantic, we should make sure that `X` is *not already bound*: that is, it doesn't already have a value, and is not a parameter of a function. Remember that Erlang allows pattern matching against bound variables: in this case the pattern match is successful only if the incoming value is equal to the value of `X`.)

Design a simple test bed to show how messages are processed in mailbox order. The simplest way to do this is to define a `receiver/0` function, which repeatedly receives messages and prints them out. You can do this by making the action on `receive` a call to `io:format/2`, for example:

```
io:format("message:~w~n",[{ok,42}])
```

You should capture the `Pid` of this process when you `spawn` it, by assigning the result of the call to `spawn` to a variable. You can then send messages to the `receiver` from the Erlang shell.

In order for the messages to have arrived in the mailbox of the `receiver` before they are processed, you can use a call to `timer:sleep(M)`, which will sleep a process for `M` milliseconds, before performing the `receive` statement.

To stop the `receiver/0` process, you'll have to interrupt the Erlang shell by typing `Ctrl-C`.

If you remove the call to `timer:sleep/1`, would you expect the behaviour of the receiver to change? Does it change when you try it out?

Stopping the process explicitly

As an alternative to using `Ctrl-C` to stop the Erlang shell, you could modify the definition of the `receiver/0` function to treat the message `stop` differently: first printing out the message, but then stopping the process.

There are two ways of doing this:

- the first will match incoming messages with a variable, and then pattern match each message using a `case` expression;

- the second will treat different cases of incoming message at the top level of the `receive` statement.

Would you expect these two to have the same behaviour? Try out the two approaches from the Erlang shell, with and without sleeping the `receiver` process before processing the messages. Do you see different behaviour at all now? How do you explain it?

Processing messages in sequence

Suppose that you want to process two messages

```
{first, FirstString}
{second, SecondString}
```

in that order. How would you define a process that will deal with these messages in that order, **irrespective of the order in which they arrive in the mailbox**. Try out your solution by spawning the process from the Erlang shell, and interacting with it.
