Prof. Dr. Annette Bieniusa
Philipp Lersch, M. Sc.

RPTU Kaiserslautern-Landau
Fachbereich Informatik
AG Softwaretechnik

Exercise 1: Programming Distributed Systems (Summer 2025)

Deadline: 7.5.2025, AOE

Information

- After you have registered for the exercise via Olat, you will get access to a new Git repository.
- In your Git repository, create a branch for this exercise sheet (for example with `git checkout -b ex1`)
- Create a folder named "ex1" in your repository and add your solutions to this folder.
- You can ask general questions in the Mattermost chat.
- Please help out other students if you know how to answer their questions, including on Mattermost - collaboration is encouraged, but do not ask how to solve an exercise or share full solutions to exercises.
- To get feedback on your solution, create a merge request into your default branch in Gitlab and assign Philipp Lersch as assignee.
- Test your submission with the provided test cases. Feel free to add more tests, but do not change the existing test cases.
- Submissions are **optional**. There will be a final project after the exercise sheets that is **mandatory** to take the exam.
- All solutions of the exercise sheets will be discussed in class. Code solutions will be provided after the exercise session. Solutions to theory questions will not be provided. Please come to the class if you are interested in the solution of theory questions.
- Exercises marked with * are hard and optional questions. They are numbered separately.

1 Exercise Team

Based on your enrollment in teams of two on OLAT you will be assigned a Gitlab repository. The link to access this repository will be send to the email you registered in OLAT. Teams consisting of only one person will be assumed to be inactive. If you do not have a teammate find one in person or on Mattermost and notify me to get late access to a Gitlab repository.

2 Characteristics of Distributed Systems

Search for an example of a distributed system and describe it in terms of the following characteristics:

Network Size How large is the network the distributed system is spanning? Local network, local area, distributed around the planet? What types of devices are involved?

Architecture What roles to the different processes take?

Resource Sharing What resources are managed by these processes?

Concurrency What type of concurrency does the system exhibit? Among servers, among users, concurrency inside one server ... ?

Transparency Is it transparent to the user that the system is distributed (e.g. Google uses the same search engine across the globe versus Amazon has different stores for each country)?

3 Tool Setup

Follow the instructions at `https://elixir-lang.org/install.html` to install the required software for the exercises.

4 Elixir Basics

Create a new Mix project: Open a terminal and navigate to the folder, where you want to create the project. Then run "`mix new ex1`". This will create a new project in the folder "ex1".

4.1 The Shell

To open a shell in your project, navigate your project folder and run "`iex -S mix`". You can use the shell to evaluate Elixir expressions, for example:

```
iex(1)> 7*6
42
iex(2)> hd [5,4,6]
5
```

You can bind the result of expressions to variables. A variable can only be assigned on the left side of the `=` operator. If you want to pattern match on the left side instead of re-assigning the variable, use the pin `^` operator.

```
iex(3)> x = 5
5
iex(4)> y = x + 1
6
iex(5)> x = 3
3
iex(6)> ^x = 3.
3
iex(7)> ^x = 4.
** (MatchError) no match of right hand side value: 4
```

The shell is useful to experiment with code you have written. To try this feature, create a new file named "`warmup.ex`" in your projects `lib`-folder. Then add the following content to the file:

```
1 defmodule Warmup do
2   def add(x, y) do
3     x + y
4   end
5 end
```

Now you can use the command `recompile` to load the new code into your shell. As the function is exported from module `Warmup`, you can now call the function from the shell to see that it can correctly add 2 numbers:

```
8> recompile
Compiling 1 file (.ex)
Generated ex2 app
:ok
9> Warmup.add 4,3
7
```

You can exit the shell with `Ctrl + g` and `q`.

4.2 Testing

Download the file `warmup_test.exs` from the course material and put it into a folder named `test` into your project. This file contains the test cases which you can use to test your solutions for this exercise. You can run them from the terminal with the command `mix test`. The results are printed on the terminal.

4.3 Numbers, Lists, and Tuples

Implement the functions described below in the module named `Warmup`. Use the provided test suite to test your implementation. Avoid using functions from the standard library and try to implement your own version of the functionality.

a) Write a function `minimum/2`, which takes two numbers and returns the smaller value of the two. Do not use the built-in `min` function.[1]

```
> Warmup.minimum(3, 7)
3
```

b) Write a function `swap/1`, which takes a pair and returns a pair where the components are swapped.

```
> Warmup.swap {:ok, 200}
{200, :ok}
```

c) Extend your function `swap/1`, to also take a triple and return a triple where the left-most and right-most components are swapped.

```
> Warmup.swap {:ok, 100, 200}
{200, 100, :ok}
> Warmup.swap {:ok, 200}
{200, :ok}
```

1*) Extend your function `swap/1` to work with any tuples of size n, where the left-most and right-most components are swapped.

d) Write a function `only_integers?/1`, which takes a list of numbers and checks, whether it contains only integers.

```
> Warmup.only_integers [4,7,5,2]
true
> Warmup.only_integers [2,4,5.0,7]
false
```

e) Write a function `delete/2`, which takes a key and a list of key-value pairs. The function should return `{:ok, l}`, where l is the new list without the given key. If no entry with the given key exists, the function should return the atom `noop`.

```
> Warmup.delete(:d, [{:c, 5}, {:z, 7}, {:d, 3}, {:a, 1}])
{:ok, [{:c, 5}, {:z, 7}, {:a, 1}]}
> Warmup.delete(:x, [{:c, 5}, {:z, 7}, {:d, 3}, {:a, 1}])
:noop
```

f) Write a function `same?/2` that takes two inputs and checks if they are structurally equivalent.

```
> Warmup.same ?{:ok, 2} {:ok, 2}
true
> Warmup.same ?{:ok, 2} {:ok, {}}
false
```

2*) Is it possible to write a function `same?/2` which checks if two inputs are the same references?

```
> Warmup.same? {1,2},{1,2}
false
> a = {1,2}
> Warmup.same? a,a
true
```

g) Given a string containing brackets `[]`, braces `{}`, parentheses `()`, or any combination thereof, verify that any and all pairs are matched and nested correctly[2].

[1] Hint: You can use the `case`-expression, `cond`-expression, `if`-expression or multiple function heads with guards: https://hexdocs.pm/elixir/case-cond-and-if.html

[2] `h(String.graphemes)` should be helpful

4.4 Higher Order Functions

h) Use `Enum.filter/2` to write a function `positive/1`, which takes a list of numbers `l` and returns a list of all numbers in `l`, which are greater or equal to 0.

```
> Warmup.positive([6, -5, 3, 0, -2])
[6, 3, 0]
```

i) Use `Enum.all?/2` to write a function `all_positive/1`, which takes a list of numbers and checks whether all numbers in the list are greater or equal to 0.

```
> Warmup.all_positive?([1, 2, 3])
true
> Warmup.all_positive?([1, -2, 3])
false
```

j) Use `Enum.map/2` to write a function `values/1`, which takes a list of key-value pairs and returns a list of only the values.

```
> Warmup.values([{:c, 5}, {:z, 7}, {:d, 3}, {:a, 1}])
[5, 7, 3, 1]
```

k) Use `List.foldl/3` to write a function `list_min`, which computes the minimal element of a nonempty list.

```
> Warmup.list_min([7, 2, 9])
2
```

5 Typing Elixir

Elixir can be typed via `@spec my_fun(integer) :: integer` annotations. One of the tools which can process these annotations is called *Dialyzer*. To configure Dialyzer properly, add it to your mix configuration[3]:

```
defp deps do
[{:dialyxir, "~> 1.0", only: [:dev, :test], runtime: false}]
end
```

and update your project dependencies with

```
mix do deps.get, deps.compile
```

Then you can check your code with `mix dialyzer`.

Examples:

```
@spec maximum(number, number) -> number.
def maximum(X,Y) do ...
```

a) Write functions that fulfill these type annotations:

```
@spec fun1(boolean, boolean) -> number
@spec fun2(list({number, number}), any) ->
{{:notmatched, number}, {:matched, number}}
@spec fun3(list(any), atom) -> list({atom, any}).
```

b) Write type annotations for all previous functions in this exercise sheet.

c) Define a precise type annotation for this function:

```
def fun4([]) do :error end
def fun4([x]) do x end
def fun4([x | [y | ys]]) do x ++ fun4([y|ys]) end
```

[3] https://hexdocs.pm/dialyxir/Mix.Tasks.Dialyzer.html

What does the function `fun4` do?