Assignment 1

Elm project - Weather app

Deadline: Sunday, December 4, 23:45

1.1 Submission instructions

1. Unzip the `Elm-Project.zip` folder. You should find (among others):
   - `src` folder - your workspace
   - `tests` folder - self evaluation tests
   - `scripts` folder - utility scripts
   - `.gitignore` - if you want to use version control
   - `elm.json` - elm project configuration
   - `package.json` - npm project configuration
2. Run `npm install` to install the dependencies needed for the automated tests.
3. Edit the source files in the `src` folder with your solutions.
4. When done, run `npm run zip` which will create a zip archive with the `src` folder.

1.2 Project resources

Table 1.1: Project Resources

Resource	Link
Elm `core` library	`https://package.elm-lang.org/packages/elm/core/1.0.5/`
Elm `html` package	`https://package.elm-lang.org/packages/elm/html/latest`
Elm `test` package	`https://package.elm-lang.org/packages/elm-explorations/test/latest/`
Elm `http` package	`https://package.elm-lang.org/packages/elm/http/latest`
Elm `json` package	`https://package.elm-lang.org/packages/elm/json/latest`

Table 1.2: Extra Resources - Talks about how to design Elm apps

Resource	Link
The life of a file - Evan Czaplicki	`https://youtu.be/XpDsk374LDE`
Making Impossible States Impossible - Richard Feldman	`https://youtu.be/IcgmSRJHu_8`
Immutable Relational Data - Richard Feldman	`https://youtu.be/28OdemxhfbU`
Make Data Structures - Richard Feldman	`https://youtu.be/x1FU3e0sT1I`

1.3 Project description, goals and non-goals

In this project you will develop basic weather app. It will show the temperature, precipitation for the next 7 days for the current location (which will be hardcoded in the app) both on plotted chart and as a list of daily high and low temperature and total precipitation.

The main goal of the project is to get hands-on experience for building a close to real-world app, that displays useful data, can retrieve data from a server and has a decent test suite to ensure that it works properly.

There are also non-goals for this project, the main one being styling - don't spend time on styling before the logic of the app is complete. Other non-goals include handling and validating more complex inputs from the user - while this use case certainly appears in the real world, it is often quite tedious and time consuming to implement and thus it is better to spend more time on simpler features that can still make a useful app.

1.4 Grading

This project is worth 30% of your final lab grade.

You can obtain in total 30 points:

- 60% (18 points) come from public tests (i.e. that you can run to check your implementation)
- 20% (6 points) come from manual grading or hidden tests (i.e that are not available to you, but will be run when grading your project)
- 20% (6 points) come from coding style

The tests will cover all functional requirements, but you can implement as much as little as you consider adequate. The grade for functional requirements will be calculated from the number of tests that pass (failing tests most likely mean that a requirement is missing or is not implemented correctly).

1.5 Getting started with the development

Starting code

Most of the logic in the `Main.elm` and `Model.elm` files is already implemented: `Main` contains the basic skeleton for the app and `Model` contains the data definitions for the model. The other files under the `Model` folder also contain some functions that are already implemented.

It is highly recommended that you spend some time to understand the existing code before starting to write your solutions.

Development process

First, you should run `npm test` to confirm that the tests fail because of `Debug.todo`. It might help to replace `Debug.todo` with a concrete value that makes the function compile, just to see that all the tests fail. Such values are placed as a comment just below the first line of the function.

Then you should start `elm reactor`, open the `src/Main.elm` file (both in reactor and in your editor) and start by commenting most of the `Main.view` function to focus on getting the other `view`s to compile. After the view you're working on compiles you can run tests to see if they pass. Then you can slowly uncomment functions in `Main.view` to repeat this procedure.

If you would like to work offline, you can use the `src/Reactor.elm` file and the `npm run server` command to use a local server to obtain some hardcoded weather data that matches the shape of the data returned by the real API.

1.6 Project tasks (functional requirements)

Exercise 1.6.1 3p + 1p

Implement the functions in the `Utils` module according to the documentation comments and examples.
These are utility functions that are needed to extract the data required by the chart and the weekly view.

Grading:
1.25p `groupBy`
1.25p `zipFilter`
1p `minimumBy` and `maximumBy`
0.5p `maybeToList`

Exercise 1.6.2 3p + 1p

Complete the `View.Day.view` function such that it shows every field of `DailyData`.
This view will show the date, minimum and maximum temperature and total precipitation for a given day.

Grading:
2p For showing all data fields (date, lowTemp, highTemp, precipitation)
0.5p The high and low temperature should have `class` *day-hightemp* and *day-lowtemp*, respectively
0.5p The precipitation should have `class` *day-precipitation*
0.5p The outer div should have `class` *day*
0.5p The date should have `class` *day-date*

Exercise 1.6.3 — 2.5p + 0.5p

Complete the `View.Week.view` function such that it shows every field of `WeeklyData`. This view will show the weather for the next week using the daily view implemented in previous exercise.

Grading:

- 1.5p The dates of the first and last day should be displayed in a `h2` element
- 1p One `View.Day.view` should be rendered for each day in `dailyData`
- 0.5p The outer div should have `class` *week*

Exercise 1.6.4 — 3p

Complete the code in the `Main` module according to the comments, examples and tests. These functions will extract the data required by the chart and weekly views from the application state.

Grading:

- 1.5p `getChartData`
- 1.5p `getWeeklyData`

Exercise 1.6.5 — 3.5p + 0.5p

Implement the functions in the `Model.WeatherData` module according to the documentation comments and examples.

Grading:

- 1.5p `decodeHourlyData`
- 1.5p `decodeWeatherData`
- 1p `toHourlyDataPoints`

Exercise 1.6.6 — 3p + 3p

Complete the `Model.WeatherItems` and `View.WeatherItems` modules according to the comments, examples and tests.
After this exercise, you should be able to change the data displayed on the chart by selecting or deselecting the corresponding checkboxes.

Grading:
For the `Model.WeatherItems` module:
- 1p Choose a suitable representation for the state that represents which categories are selected, then complete `SelectedWeatherItems` with the definition (it can be a type or a type alias).
- 1p Implement the `allSelected` and `isItemSelected` functions.
- 1p Implement the `set` function.

For the `View.WeatherItems` module:
- 1p Implement the `view` function that shows 4 checkboxes using the given `checkbox` function.

You'll also have to make some modifications in the `Main` module:
- 1p Complete the `Main.update` function to handle when an event category is selected or deselected
- 1p Complete the `Main.view` function to use `View.WeatherItems.view`

Hint: You might get some inspiration from the extra resources.

1.7 Coding style (non-functional requirements)

Exercise 1.7.1 — 3p

Properly use Elm language features and library functions. Examples include:
- 1p Lambda functions
- 1p Pipelines and function composition
- 1p Built-in functions:
  - Functions for list processing (`List.map`, `List.filter`, `List.foldl`, etc.)
  - Functions for error handling (`Maybe.map`, `Maybe.withDefault`, etc.)

Exercise 1.7.2 — 3p

Use a proper coding style:
- 1.5p Descriptive names for data definitions and functions
- 1.5p Readable code structure (proper use of indentation)

1.8 Testing your implementation

The project contains both traditional test that can be run with `elm-test` and examples that can be run with `elm-verify-examples`. You have to `npm install` (once) to run tests.

To run all test and see your final grade, use:

```powershell session
PS > npm run grade
```

To run all test and see detailed explanations for all failures, use:

```powershell session
PS > npm test
```

To run all tests manually, you can use:

```powershell session
PS > npx elm-verify-examples; npx elm-test
```