Assignment 1

Elm project

Deadline: Sunday, November 27, 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. Edit the source files in the `src` folder with your solutions.
3. Run `npm install` to install the dependencies needed for the automated tests.
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 app to showcase your portfolio: education, work, projects and awards/achievements. The page will start with contact details. Then, the events will be show in a timeline that includes all events sorted in chronological order. The user (visitors of your site) will be able to select which event categories to show in the timeline. Finally a list of your top projects sorted by stars on github.

If you are pleased with the result of your project, you will be able to easily publish it using github pages.

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 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 and some sample data. 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.

1.6 Project tasks (functional requirements)

Exercise 1.6.1 3p

Complete the `Model.PersonalDetails.view` function such that it shows every field.

Grading:
- 0.5p The name should be in a *h1* tag and `id` *name*
- 0.5p The intro should be in a *em* tag and `id` *intro*
- 1p Each contact detail should have `class` *contact-detail*
- 1p Each social media link should have `class` *social-link* and use the `a` tag with a `href` attribute for the links

Exercise 1.6.2 3p

Complete the `Model.Event.view` function such that it shows every field.

Grading:
- 0.5p Events should have `class` *event*
- 0.5p The title should have `class` *event-title*
- 0.5p The description should have `class` *event-description*
- 0.5p The category should have `class` *event-category*
- 0.5p The url should have `class` *event-url*
- 0.5p If the `important` field is `True`, the event should have `class` *event-important*

Exercise 1.6.3 — 6p

Complete the functions in `Model.Interval`, `Model.Date` and `Model.Event` modules according to the comments, examples and tests.

Grading:

- `Model.Date`:
  - 0.5p `Date.monthsBetweenMonths`
  - 0.5p `Date.compare` (hint: use the `Model.Util.chainCompare function`)
  - 1p `Date.monthsBetween`
  - 1p `Date.view`:
    - The final view should contain the year and month (as given by `monthToString`), if present, in the format of your choice
- `Model.Interval`:
  - 1p `Interval.compare` (hint: use the `Model.Util.chainCompare` function)
  - 1.5p `Interval.view`:
    - Intervals should have `class` *interval*
    - The start date should have `class` *interval-start*
    - The end date should have `class` *interval-end*. If the end date is missing, the text "Present" should be used.
    - The `start` and `end` fields **must** be in separate `text` nodes!
    - If it can be calculated (i.e. `Interval.length` function returns `Just`), display the length of the interval in years and months using the `lenght` function. This field should have `class` *interval-length*.
- `Model.Event`:
  - 0.5p `sortByIntervals`

Exercise 1.6.4 — 3p

Complete the `Model.Repo` module according to the comments, examples and tests:

Grading:

- 0.5p Implement the `decodeRepo` function.
- 0.5p Implement the `sortByStars` function.
- 1p Make the necessary modifications in the `Main` module to fetch the repositories when the app is started:
  - Complete the `Main.init` function to fetch the repositories when the app is started.
  - Complete the `Main.update` function to add the fetched repositories to the model.
- 1p Implement the `view` function:
  - Repos should have `class` *repo*
  - The name should have `class` *repo-name*
  - The description should have `class` *repo-description*
  - The url should have `class` *repo-url*, and should contain an `a` tag with a `href` attribute that links to the repo
  - The stars should have `class` *repo-stars*

Exercise 1.6.5 3p

Complete the `Model.Event.Category` module according to the comments, examples and tests:

Grading:

- 1p Choose a suitable representation for the state that represents which categories are selected, then complete `SelectedEventCategories` with the definition.
- 0.5p Implement the `allSelected` and `isEventCategorySelected` functions.
- 0.5p Implement the `set` function.
- 0.5p Implement the `view` function that shows 4 checkboxes using the given `checkbox` function.
- 0.5p Make the necessary modifications in the `Main` module:
  - Complete the `Main.update` function to handle when an event category is selected or deselected

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 Pipelines
- 1p Lambda functions
- 0.5p Function composition
- 0.5p 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 grade, use:

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

To run all tests manually, you can use:

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