Elm Friendly functional frontend

CONCEPTS

Read compiler errors top to bottom: they usually name the fix.

Model all state
↓ view
View Html Msg
↓ user event
Msg what happened
↓ update
Update returns new Model
The Elm Architecture

PROJECT

`elm init`
Create elm.json and src/

`elm install <author/package>`
Add a dependency

`elm repl`
Start an interactive session

BUILD

`elm make src/Main.elm`
Compile to index.html

`elm make src/Main.elm --output=elm.js`
Compile to a JS file

`elm make src/Main.elm --optimize`
Optimized build, no Debug

`elm reactor`
Serve the project at :8000

PACKAGES

`elm install elm/json`
JSON encode and decode

TOOLING

`elm-format src/ --yes`
Format all files in place

`elm-test init`
Create tests/ and test deps

TOOLING (cont.)

`elm-test`
Run all tests in tests/

SYNTAX

```
module Main exposing (main)
```
Module header

```
import Html exposing (Html, div, text)
```
Import names

```
name : Int → String
```
Type annotation

```
type alias User = { name : String }
```
Name a record type

```
type Msg = Inc | Dec
```
Custom type

```
case x of ... → ...
```
Pattern match

```
{ model | count = 1 }
```
Update record fields

```
x ▷ f ▷ g
```
Pipe left to right

CORE LIBRARY

`List.map f xs`
Transform each item

`List.filter f xs`
Keep matching items

`String.fromInt n / String.toInt s`
Int to string / parse (Maybe)

`Maybe.withDefault 0 m`
Unwrap Maybe with fallback

PROGRAMS

`Browser.sandbox`
Simple app: init, view, update

`Browser.element`
App with Cmd, Sub and flags

`Html.Events.onClick Msg`
Send a Msg on click