WIKIPEDIA

Elm (programming language)

Elm is a domain-specific programming language for declaratively creating web browser-based graphical user interfaces. Elm is purely functional, and is developed with emphasis on usability, performance, and robustness. It advertises "no runtime exceptions in practice",[6] made possible by the Elm compiler's static type checking.

Elm

Paradigm	Functional
Designed by	Evan Czaplicki
First appeared	March 30, 2012[1]
Stable release	0.19.1 / October 21, 2019[2]
Typing discipline	Static, Strong, Inferred
License	Permissive (Revised BSD)[3]
Filename extensions	.elm
Website	elm-lang.org (https://elm-lang.org/)

Influenced by

Haskell, Standard ML, OCaml, F#

Influenced

Redux,[4] Vue[5]

Contents

History

Elm was initially designed by Evan Czaplicki as his thesis in 2012.[7] The first release of Elm came with many examples and an online editor that made it easy to try out in a web browser.[8] Evan joined Prezi in 2013 to work on Elm,[9] and in 2016 moved to NoRedInk as an Open Source Engineer, also starting the Elm Software Foundation.[10]

The initial implementation of the Elm compiler targets HTML, CSS, and JavaScript.[11] The set of core tools has continued to expand, now including a REPL,[12] package manager,[13] time-travelling debugger,[14] and installers for macOS and Windows.[15] Elm also has an ecosystem of community created libraries (https://package.elm-lang.org/packages/) and Ellie (https://ellie-app.com), an advanced online editor that allows saved work and inclusion of community libraries.

Features

Elm has a small set of language constructs, including traditional if-expressions, let-expressions for local state, and case-expressions for pattern matching.[16] As a functional language, it supports anonymous functions, functions as argumensts, and functions can return functions, the later often by partial application of curried functions. Functions are called by value. Its semantics include immutable values, stateless functions, and static typing with type inference. Elm programs render HTML through a virtual DOM, and may interoperate with other code by using "JavaScript as a service".

Immutability

All values in Elm are immutable, meaning that a value cannot be modified after it is created. Elm uses persistent data structures to implement its Arrays, Sets, and Dictionaries in the standard library.[17]

Static types

Elm is statically typed. Type annotations are optional (due to type inference) but strongly encouraged. Annotations exist on the line above the definition (unlike C-family languages where types and names are interspersed). Elm uses a single colon to mean "has type".

Types include primitives like integers and strings, and basic data structures such as lists, tuples, and records. Functions have types written with arrows, for example `round : Float -> Int`. Custom types allow the programmer to create custom types to represent data in a way that matches the problem domain.[18]

Types can refer to other types, for example a `List Int`. Types are always capitalized; lowercase names are type variables. For example, a `List a` is a list of values of unknown type. It is the type of the empty list and of the argument to `List.length`, which is agnostic to the list's elements. There are a few special types that programmers create to interact with the Elm runtime. For example, `Html Msg` represents a (virtual) DOM tree whose event handlers all produce messages of type `Msg`.

Rather than allow any value to be implicitly nullable (such a JavaScript's `undefined` or a null pointer), Elm's standard library defines a `Maybe a` type. Code that produces or handles an optional value does so explicitly using this type, and all other code is guaranteed a value of the claimed type is actually present.

Elm provides a limited number of built-in type classes: `number` which includes `Int` and `Float` to facilitate the use of numeric operators such as `(+)` or `(*)`, `comparable` which includes numbers, characters, strings, lists of comparable things, and tuples of comparable things to facilitate the use of comparison operators, and `appendable` which includes strings and lists to facilitate concatenation with `(++)`. Elm does not provide a mechanism to include custom types into these type classes or create new type classes (see Limitations section).

Module system

Elm has a module system that allows users to break their code into smaller parts called modules. Modules can hide implementation details such as helper functions, and group related code together. Modules serve as a namespace for imported code, such as `Bitwise.and`. Third party libraries (or packages) consist of one or more modules, and are available from the Elm Public Library (https://package.elm-lang.org/). All libraries are versioned according to semver, which is enforced by the compiler and other tools. That is, removing a function or changing its type can only be done in a major release.

Interoperability with HTML, CSS, and JavaScript

Elm uses an abstraction called ports to communicate with JavaScript.[19] It allows values to flow in and out of Elm programs, making it possible to communicate between Elm and JavaScript.

Elm has a library called elm/html that a programmer can use to write HTML and CSS within Elm.[20] It uses a virtual DOM approach to make updates efficient.[21]

Backend

Elm does not officially support server-side development. The core development team does not consider it as their primary goal and prefers to focus development on the enhancement of front-end development experience. Nevertheless, there are several independent projects, which attempt to explore possibilities to use Elm for the back-end. The projects are mainly stuck on Elm version 0.18.0 since newer ones do not support "native" code and some other utilized features. There are two attempts to use Elm with BEAM (Erlang virtual machine). One of the projects executes Elm directly on the environment[22] while another one compiles it to Elixir.[23] Also, there was an attempt to create a back-end framework for Elm powered by Node.js infrastructure.[24] None of the projects are production-ready.

The Elm Architecture

The Elm Architecture is a pattern for building interactive web applications. Elm applications are naturally constructed in that way, but other projects may find the concept useful.

An Elm program is always split into three parts:

- Model - the state of the application
- View - a function that turns the model into HTML
- Update - a function that updates the model based on messages

Those are the core of the Elm Architecture.

For example, imagine an application that displays a number and a button that increments the number when pressed.[25] In this case, all we need to store is one number, so our model can be as simple as `type alias Model = Int`. The `view` function would be defined with the `Html` library and display the number and button. For the number to be updated, we need to be able to send a message to the `update` function, which is done through a custom type such as `type Msg = Increase`. The `Increase` value is attached it to the button defined in the `view` function such that when the button is clicked by a user, `Increase` is passed on to the `update` function, which can update the model by increasing the number.

In the Elm Architecture, sending messages to `update` is the only way to change the state. In more sophisticated applications, messages may come from various sources: user interaction, initialization of the model, internal calls from `update`, subscriptions to external events (window resize, system clock, JavaScript interop...) and URL changes and requests.

Limitations

Elm does not support higher-kinded polymorphism,[26] which related languages Haskell and PureScript offer, nor does Elm support the creation of type classes.

This means that, for example, Elm does not have a generic `map` function which works across multiple data structures such as `List` and `Set`. In Elm, such functions are typically invoked qualified by their module name, for example calling `List.map` and `Set.map`. In Haskell or PureScript, there would be only one function `map`. This is a known feature request that is on Czaplicki's rough roadmap since at least 2015.[27]

Another outcome is a large amount of boilerplate code in medium to large size projects as illustrated by the author of "Elm in Action" in their single page application example[28] with almost identical fragments being repeated in update, view, subscriptions, route parsing and building functions.

Example code

```
-- This is a single line comment.

{-
This is a multi-line comment.
It is {- nestable. -}
-}

-- Here we define a value named `greeting`. The type is inferred as a `String`.
greeting =
    "Hello World!"

 -- It is best to add type annotations to top-level declarations.
hello : String
hello =
    "Hi there."

-- Functions are declared the same way, with arguments following the function name.
add x y =
    x + y

-- Again, it is best to add type annotations.
hypotenuse : Float -> Float -> Float
hypotenuse a b =
    sqrt (a^2 + b^2)

-- We can create lambda functions with the `\[arg] -> [expression]` syntax.
hello : String -> String
hello = \s -> "Hi, " ++ s

-- Function declarations may have the anonymous parameter names denoted by `_`, which are
matched but not used in the body.
const : a -> b -> a
const k _ = k


-- Functions are also curried; here we've curried the multiplication
-- infix operator with a `2`
multiplyBy2 : number -> number
multiplyBy2 =
    (*) 2

-- If-expressions are used to branch on `Bool` values
absoluteValue : number -> number
absoluteValue number =
    if number < 0 then negate number else number

 -- Records are used to hold values with named fields
book : { title : String, author : String, pages : Int }
book =
    { title = "Steppenwolf"
    , author = "Hesse"
    , pages = 237
    }
```

```
-- Record access is done with `.`
title : String
title =
    book.title

-- Record access `.` can also be used as a function
author : String
author =
    .author book

-- We can create tagged unions with the `type` keyword.
-- The following value represents a binary tree.
type Tree a
    = Empty
    | Node a (Tree a) (Tree a)

-- It is possible to inspect these types with case-expressions.
depth : Tree a -> Int
depth tree =
    case tree of
        Empty ->  0
        Node _ left right ->
            1 + max (depth left) (depth right)
```

See also

- PureScript: a strongly-typed, purely-functional programming language that compiles to JavaScript
- Reason: A syntax extension and toolchain for OCaml that can also transpile to JavaScript

References

1. Czaplicki, Evan (30 March 2012). "My Thesis is Finally Complete! "Elm: Concurrent FRP for functional GUIs" " (https://www.reddit.com/r/haskell/comments/rkyoa/my_thesis_is_finally_complete_elm_concurrent_frp/). *Reddit*.
2. "Releases · elm/Compiler" (https://github.com/elm/compiler/releases).
3. "elm/compiler" (https://github.com/elm/compiler/blob/master/LICENSE). *GitHub*. 16 October 2021.
4. "Prior Art - Redux" (https://redux.js.org/introduction/prior-art). *redux.js.org*.
5. "Comparison with Other Frameworks — Vue.js" (https://vuejs.org/v2/guide/comparison.html#Scale).
6. "Elm home page" (https://elm-lang.org/).
7. "Elm: Concurrent FRP for Functional GUIs" (https://elm-lang.org/assets/papers/concurrent-frp.pdf) (PDF).
8. "Try Elm" (https://web.archive.org/web/20170521144831/http://elm-lang.org/try). *elm-lang.org*. Archived from the original (https://elm-lang.org/try) on 2017-05-21. Retrieved 2019-07-24.
9. "elm and prezi" (https://elm-lang.org/news/elm-and-prezi). *elm-lang.org*.
10. "new adventures for elm" (https://elm-lang.org/news/new-adventures-for-elm). *elm-lang.org*.
11. "elm/compiler" (https://github.com/elm/compiler). *GitHub*. 16 October 2021.
12. "repl" (https://elm-lang.org/news/repl). *elm-lang.org*.
13. "package manager" (https://elm-lang.org/news/package-manager). *elm-lang.org*.
14. "Home" (https://elm-lang.org/news/time-travel-made-easy). *elm-lang.org*.
15. "Install" (https://guide.elm-lang.org/install.html). *guide.elm-lang.org*.

16. "syntax" (https://web.archive.org/web/20160313052210/http://elm-lang.org/learn/syntax.elm). *elm-lang.org*. Archived from the original (http://elm-lang.org/learn/Syntax.elm) on 2016-03-13. Retrieved 2013-05-31.
17. "elm/core" (https://package.elm-lang.org/packages/elm/core/latest/). *package.elm-lang.org*.
18. "Model The Problem" (https://guide.elm-lang.org/types/custom_types.html). *Elm*. Retrieved 4 May 2016.
19. "JavaScript interop" (https://guide.elm-lang.org/interop/). *elm-lang.org*.
20. "elm/html" (https://package.elm-lang.org/packages/elm/html/latest/). *package.elm-lang.org*.
21. "Blazing Fast HTML" (https://elm-lang.org/news/blazing-fast-html). *elm-lang.org*.
22. "Kofigumbs/Elm-beam" (https://github.com/hkgumbs/elm-beam). 24 September 2021.
23. "What is it?" (https://github.com/wende/elchemy). 24 September 2021.
24. "Board" (https://github.com/AIRTucha/board). 18 September 2021.
25. "Buttons · An Introduction to Elm" (https://guide.elm-lang.org/architecture/buttons.html). *guide.elm-lang.org*. Retrieved 2020-10-15.
26. "Higher-Kinded types Not Expressible? #396" (https://github.com/elm-lang/elm-compiler/issues/396). *github.com/elm-lang/elm-compiler*. Retrieved 6 March 2015.
27. "Higher-Kinded types Not Expressible #396" (https://github.com/elm/compiler/issues/396#issuecomment-128190898). *github.com/elm-lang/elm-compiler*. Retrieved 19 November 2019.
28. "Main.elm" (https://github.com/rtfeldman/elm-spa-example/blob/23dee34dd7a8c26229a03bc8e9f0e034f6222f13/src/Main.elm). *github.com/rtfeldman/elm-spa-example*. Retrieved 30 June 2020.

External links

- Official website (https://elm-lang.org/)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Elm_(programming_language)&oldid=1056731864"

This page was last edited on 23 November 2021, at 06:47 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.