U60SXAF96 : Thanks.
U60SXAF96 : I appreciate it.
U3LUC6SNS : <@U48AEBJQ3>, I also ran into the jumping cursor bug. I'm using workaround that is adapted from various online resources including this slack channel: <https://github.com/jxxcarlson/nanoedit>
I think I had an Ellie on this. Is there a way to search Ellies?

U3LUC6SNS : If you search the style-elements channel for "jumping", you will find some discussion of this. The use of `counter` or a better substitute like a document ID is essential to make the virtual DOM comply with your wishes.
U3KSN5MAL : <@U3SJEDR96> wait, how can i do toUpper on the strings if the decoder is being called by the test?
U48AEBJQ3 : <@U3KSN5MAL> I would suggest writing `stringsToUpper : List String -&gt; List String` then look for a way to use that function in your decoder.
U611WQPL4 : &gt; In the tests, there is no field-name for the first one. The value is literally _just_ `5``Literally.Literally 5`

U3KSN5MAL : ok thanks
U0JFGGZS6 : more on jumping cursor -&gt; <https://github.com/elm-lang/html/issues/105>
U0JFGGZS6 : and <https://github.com/elm-lang/html/issues/55>
U3KSN5MAL : ```stringsToUpper : List String -&gt; List StringstringsToUpper list =
    let
        up t =
            toUpper t
    in
        List.map up list

decoder : Decoder (List String)
decoder =
    Json.Decode.map stringsToUpper (list decodeString)```

U3KSN5MAL : ok can't work out what i'm doing wrong :confused:
U5VTA57UN : <@U3KSN5MAL> `up = (\t -&gt; toUpper t)` ?
U3KSN5MAL : i know that
U3KSN5MAL : i just write verbose and compress later as it's easier for me
U0LPMPL2U : `up` and `toUpper` are the same right?
U48AEBJQ3 : `decodeString` isn't a `Decoder`, you want just `string`
U3KSN5MAL : doh
U3KSN5MAL : that was it
U3KSN5MAL : thanks
U3KSN5MAL : and yes you are correct joel just silly mistakes
U3KSN5MAL : thanks
U3KSN5MAL : God i'd hate to think about how messy my code base would be for anyone else to look at -_-
U3KSN5MAL : Might have to go through and do a big semantic compression pass once i get this update out
U48AEBJQ3 : <@U3KSN5MAL> One nice thing about Elm is that it feels so much safer to do major refactors, so one *can* write a bunch of messy, ugly code, then go back and clean it up and have few problems.
U3KSN5MAL : Oh yeah of course
U3KSN5MAL : i've already refactored a lot of things before
U3KSN5MAL : Like i did a total overhaul of the colour system which everything relies on halfway through
U3KSN5MAL : The biggest thing that will take work to refactor is the monolithic update loop
U48AEBJQ3 : That is something which gets easier with experience.
U3KSN5MAL : I've gone to do that and given up 3 times so far -_-
U3KSN5MAL : and i just keep adding features making the eventuallity worse
U60SXAF96 : Is there a good, pure URL validation library for Elm?
U3HQVHERX : what do you mean by url validation?
U60SXAF96 : Just something so that I can refer to a `URL` type and have confidence that it's sane, as oppose to an arbitrary `String`.
U3HQVHERX : take a look at the url-parser and navigation libraries
U60SXAF96 : Equivalent to the `URL` class in Java, or `URI.js`.
U3HQVHERX : it gives you a record from `window.location`
U60SXAF96 : Not quite what I was looking for, but very useful for my next problem. :smile:
U5XC2FJ1Y : does elm have support for pattern matching with conditionals?

U300LJUAK : <@U5XC2FJ1Y> No. At least not for now.
U23SA861Y : so, if I install via NPM i get platform version 15, anyone know whats up with that?
U300LJUAK : Gotta admit that's a feature I would love too.
U5XC2FJ1Y : what's the best alternative, just moving the conditional inside the matched pattern?
U300LJUAK : Yup. Although it can lead to duplicated code in your `else` case, that's basically the only way to go right now.
U2FP79HN3 : How do recursive types work? Say I have a cell which can be linked to other cells..
```
type alias Cell =
    { row : Int
    , column : Int
    , links : List Cell
    }
```

doesn't work, so I tried

```
type alias Cell =
    { row : Int
    , column : Int
    , links : Links
    }


type Links
    = List Cell
```

which didn't really work with:

```
link : Cell -&gt; Cell -&gt; Cell
link cell neighbour =
    { cell | links = cell.links :: neighbour }
```

and then I tried

```
type Links
    = Links (List Cell)
```

But now I'm in type un/wrapping hell

U0JFXEUCT : I believe you want something like `type Cell = Cell {}`
U0JFXEUCT : instead of a type alias
U0JFXEUCT : There is still some unwrapping, but remember you can unwrap in the function arguments
U0JFXEUCT : something like `link (Cell cell) = --do stuff`
U0CLDU8UB : The compiler suggests something like that to you when you make a recursive type alias! :slightly_smiling_face:
U2FP79HN3 : Yeah, I've read <https://github.com/elm-lang/elm-compiler/blob/0.18.0/hints/recursive-alias.md> but still confused
U0CLDU8UB : Okay, so reiterating what Matt said, you can do this:```
```
type Cell =
    Cell
    { row : Int
    , column : Int
    , links : List Cell
```

```
    }
```

U0CLDU8UB : and then something like```
link : Cell -&gt; Cell -&gt; Cell
link (Cell cell) neighbour =
   Cell { cell | links = cell.links :: neighbour }
```

U0LPMPL2U : If you find yourself unwrapping, doing something with the data, and re-wrapping a lot, I find it helpful to define a `map` function
```