U3SJEDR96 : Yeah, that's alright, it's hard to explain because intuitively, what you see is a reflection of your state :slightly_smiling_face:
U5W5F6QGP : I'm rewritting a react/redux production app in Elm atm and have some cognitive dissonance about my approach
The main subject is a `List Assessment`, where an Assessment has ~200 points of data (varying types) that are entered

My gut says to go with the obvious approach and have a giant `Assessment` record with `Maybe` values, and for updating I was thinking of a generic `UpdateAssessment String a` that targets a specific record member and performances a merge. This is almost a 1-to-1 with the current redux implementation.

I am slightly nervous about how that would play out in practice though, any advice for big records?

U3SJEDR96 : I'd go for a `Dict String Data`, I think
U3SJEDR96 : though you'll probably have some keys where some data doesn't make sense for
U3SJEDR96 : :thinking_face:
U5W5F6QGP : I COULD subdivide data by ~5 (they're laid out into fairly loose categories)The nesting seems like it would introduce more complexity though

U57KYFW67 : If the values vary, and you know all the possible things it could be, you want to use a union type most likely.
U57KYFW67 : rather than lots of maybe values
U3SJEDR96 : possibly, a mix of a record with some strongly typed keys, and a `Dict String Data` for the "freeform" stuff
U5W5F6QGP : It's one of those things where it's TextField/Checkbox/Multi-checkbox/Dropdown/TextareaAlways going to be annoying to model out data like that I guess

Thanks y'all! I'll trial a couple of small scale things and see how they play out

U3CUFAX4H : is there a sugar syntax for:```
case thingThatReturnsResult of
  Ok thing ->
    thing
  Err _ ->
    handleErrCaseThatCanNeverOccur
```

U3CUFAX4H : ?
U3SJEDR96 : _never_ is a big word. If it is truly impossible, either return a default (`thingThatReturnsResult |&gt; Result.withDefault "fallback"`) or crash.
U57KYFW67 : <@U3CUFAX4H> I wish there was a nicer way to handle that kind of thing, but you can consider it a bit of a feature that there isn't.
U57KYFW67 : What you have is an ugly solution, but it also signals that there IS that possibility (or at least you haven't disprove it in the type system)
U0CLDU8UB : Yeah, I was going to ask what sort of sugar you're referring to in Elm?
U57KYFW67 : I often pepper in `Debug.crash "IMPOSSIBLE"` in that kind of situation, though.
U0CLDU8UB : Since in my opinion Maybe doesn't have any sugar either.
U3CUFAX4H : syntactic sugar, I understand that potentially handling error cases is important for the type system in this case but I am generating the string list that is being parsed so no error can occur (if the code is correct, of course)
U57KYFW67 : You could do
U4872964V : why are you generating a string list to parse?
U3CUFAX4H : why do you always need to know what I'm doing with code I want to write? hahaha :smile:
U57KYFW67 : `Result.withDefault (Debug.crash "IMPOSSIBLE") thingThatReturnsResult` as well
U4872964V : because therein often lies the solution :slightly_smiling_face:
U57KYFW67 : (tbh, I wish there was a crash that didn't require passing a `String` message...)
U0CLDU8UB : Right. So I would maybe try incorporating the generation and the parsing (why do you need to make that into a String in the meantime?) into a single function so that you can never call the function with a non-working string.
U0CLDU8UB : Then use Debug.crash and write fuzz tests.
U3CUFAX4H : Sometimes code is written just because for me....
U0CLDU8UB : in that case you'd still have to pass `()` or similar
U3SJEDR96 : makes me wonder how many `crash`'s you have in your code o_O
```

U0CLDU8UB : true.
U3CUFAX4H : @tic-tacs, right?! I really wanted to get a custom return for handling decoder errors and spent two day fidling with that, unfortunately I am not as good a programmer as I'd like yet.
U3SJEDR96 : What do you mean by that, exactly?
U0CLDU8UB : There is one in my hobby game project, and none in the three real world customer projects I've worked on.
U3CUFAX4H : decoders are a great use case for not having a string return
U4872964V : There is always `""` if you are so inclined
U3CUFAX4H : the underlying java script that handle the Result Error case
U3CUFAX4H : but my JS is poor
U3SJEDR96 : I mean, what's the goal? Stronger error types? Or...?
U57KYFW67 : <@U3SJEDR96> At least Elm doesn't allow Haskell's one-case pattern matching lets....
U57KYFW67 : I have some old code for my IRC bot which, if another human ever laid eyes on it, I would never be allowed to program ever again....
U3CUFAX4H : Passing which ever err type you want to return to the function type declaration and having that be returned, in the same way that you should provide the string in case of an err:```
case thingy thing of
  Ok properGood ->
    do some processing returning 'Ok whatever'
  Err msg ->
    Err (TypeToReturnAsErr contentOfErrorThing)
```

U3SJEDR96 : this is out of curiosity, really :slightly_smiling_face: I've played around with decoders a lot, and experimented with stronger error types (<http://package.elm-lang.org/packages/zwilias/json-decode-exploration/3.0.0/Json-Decode-Exploration>) and a simpler version of that is also coming to Elm in 0.19
U57KYFW67 : That looks like you could probably make use of `Result.map`
U3SJEDR96 : Oh, right. I could actually add a `failWith : Error -&gt; Decoder a` option :thinking_face:
U57KYFW67 : `Result.map` allows you to transform data contained inside a result.... if there is any.... otherwise, it keeps the error the same
U0CLDU8UB : <https://github.com/ohanhi/ohanhi.com/blob/master/site.hs#L105-L110> :grimacing:
U3CUFAX4H : <@U3SJEDR96>, isn't that what the null decoder does?
U3CUFAX4H : (kinda)
U3SJEDR96 : If you're using `null ERROR` and checking on `Ok ERROR`... Maybe? But... Ew :scream:
U3SJEDR96 : considering that sends cold shivers down my spine
```