U0EUHKVGB : Error messages are worse as a result.
U0EUHKVGB : So, for example:
U0EUHKVGB : ```changeSlideOption : Bool -&gt; { rgbo : Bool, rgbl : Bool, hsvo : Bool , hsvl : Bool } -&gt; { rgbo : Bool, rgbl : Bool, hsvo : Bool , hsvl : Bool }
```

U0EUHKVGB : this is unreadable.
U0EUHKVGB : ```type alias SliderOptions =
  { rgbo : Bool
        , rgbl : Bool
        , hsvo : Bool
        , hsvl : Bool
        }

changeSlideOption : Bool -&gt; SliderOptions -&gt; SliderOptions
```
this is easy to understand.

U3KSN5MAL : Ok, i getcha. For my specific usecase with these, it worked out to be fine
U3KSN5MAL : I only did it like this because they are only flags that are only ever directly accessed in update
U0EUHKVGB : Decoding in Elm for records is generally reliant on having a named type aliases. If you want to write a decoder easily for `SliderOptions`, you would need to create a type alias so that it generates the `SliderOptions` constructor.
U3KSN5MAL : and i never had to write any functions on them
U3KSN5MAL : but it's easy enough to change
U0EUHKVGB : ```type alias ConvertedModel =
    { sliderOptions : SliderOptions
    }
```
is easier to read

U0EUHKVGB : And that means that the type error you get if you try to access the wrong field will be better
U3KSN5MAL : My code base is already at about 4.5K lines long so i think i was looking for ways to avoid verbosity
U3KSN5MAL : Anyways man i get it :slightly_smiling_face:
U0EUHKVGB : i.e `SliderOptions has no field "dfghjk"` rather than `{ rgbo : Bool, rgbl : Bool, hsvo : Bool , hsvl : Bool } has no field "dfghjk"`
U0EUHKVGB : Basically, by avoiding making a type alias you're losing out on some the aid that Elm can provide for you :slightly_smiling_face:
U3KSN5MAL : Thanks
U3LUC6SNS : <@U3KSN5MAL>  re decoders, also this: <http://eeue56.github.io/json-to-elm/>
U0EUHKVGB : <@U3LUC6SNS> That's where the conversation arose from :slightly_smiling_face:
U3LUC6SNS : I have a search function in the app I'm working.  __After__ a search, I need to do a few things: (1) select the first document in the list of search hits, (2) possible change the app page to display those results. That is, I have to sequence actions.  How do I do this?
U3KSN5MAL : yeah we are talking about bugs in it lol
U3SJEDR96 : <@U3LUC6SNS> selecting the first document sounds like something that is part of your model? changing the app page too, though, but there you might also want to update the url.. As for sequencing actions - if they can be modelled as tasks, you can use `Task.andThen` to sequence them
U48AEBJQ3 : <@U3LUC6SNS> I picture it something like.```
type alias Model =
    { searchResults : List Document
    , displayDocument : Maybe Document
    }

update msg model =
    case msg of
        PerformSearch query -&gt;
            let
                results =
                    searchFn query model.documents
```

```
        maybeDoc =
            List.head results |&gt; Maybe.andThen (\doc -&gt; if wantToDisplay doc then Just doc else Nothing)
    in
        ( { model | searchResults = results, displayDocument = maybeDoc } )
```

U3LUC6SNS : <@U48AEBJQ3> Thanks very much! I will try that approach. Nice pseudocode!!
U0JFGGZS6 : The question is does displaying the results mean changing the URL?
U3LUC6SNS : I don't have to change a URL since I am not using navigation. The part that was missing in my thinking about this was Maybe.andThen ... I will give that a try
U0JFGGZS6 : right
U48AEBJQ3 : <@U3LUC6SNS> I did it with the `andThen` because I think it teaches a more general pattern, but in production code I would probably extract out that functionality into a function. Luckily, it has already been done via```
    List.head results |&gt; Maybe.Extra.filter wantToDisplay

U3KSN5MAL : trying to modify this tuple decoder to work with 3 values not 2, my attempt just failed.
```arrayAsTuple2 : Decoder a -&gt; Decoder b -&gt; Decoder (a, b)
arrayAsTuple2 a b =
    index 0 a
        |&gt; andThen (\aVal -&gt; index 1 b
        |&gt; andThen (\bVal -&gt; Json.succeed (aVal, bVal)))```

U3KSN5MAL : my attempt
```
arrayAsTuple3 : Decoder a -&gt; Decoder b -&gt; Decoder c -&gt; Decoder (a, b, c)
arrayAsTuple3 a b c =
    Json.index 0 a
        |&gt; andThen
            (\aVal -&gt;
                Json.index 1 b
                    |&gt; andThen
                        (\bVal -&gt;
                            Json.index 2 c
                                |&gt; andThen (\cVal -&gt; Json.succeed (aVal, bVal, cVal))
)
)```

U3SJEDR96 : `map2 (,) (index 0 a) (index 1 b)`
U3SJEDR96 : or `map3 (,,) decoder1 decoder2 decoder3` for a tuple with 3 members. or `decode (,) |&gt; ....` if you want to use the pipeline stuff