U0EUHKVGB : It's not hard to access - it's hard to understand.
U3KSN5MAL : oh do you mean the name is bad?
U0EUHKVGB : Imagine you want to have a function that takes `rgbo` and returns a new `rgbo`
U0EUHKVGB : Well, the name might be bad, but that's not my point
U3KSN5MAL : hmm, these are just flags
U3KSN5MAL : not data that will be transformed
U0EUHKVGB : ```type alias ConvertedModel =
  { sliderOptions :
    { rgbo : Bool
    , rgbl : Bool
    , hsvo : Bool
    , hsvl : Bool
    }
  }
```

U0EUHKVGB : If you want to do _anything_ with `sliderOptions`, then you have to write the full type out again.
U0EUHKVGB : Error messages are worse as a result.
U0EUHKVGB : So, for example:
U0EUHKVGB : ```changeSlideOption : Bool -&amp;gt; { rgbo : Bool, rgbl : Bool, hsvo : Bool , hsvl : Bool } -&amp;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 -> SliderOptions -> 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: