U4RR7KX45 : so it has to be explicit
U4RR7KX45 : yeah thought so
U3SJEDR96 : You can't prove at compile-time that your record will have a field with that name, and that changing the value will result in the correct record shape being returned
U4RR7KX45 : yeah makes sense elm-way :smile:
U4RR7KX45 : thank you
U4872964V : If you have things like that, maybe a `Dict` is what you are after
U5E99RPK6 : If the order of keys in a record doesn't matter, how can `Model` be used as a constructor function [here](<https://guide.elm-lang.org/architecture/user_input/forms.html>)? How would it know which argument goes with which key?
U5R6L5MT4 : Thanks to you I know have a working version :slightly_smiling_face:
U5R6L5MT4 : Also it would benefit of some refactoring
U5R6L5MT4 : At least it works :slightly_smiling_face:
U2GPAEU1L : <@U5E99RPK6>
The order doesn't matter if defining the alias with `{ }` syntax, so:

`{ a = 5, b= 6 }` or `{ b = 6, a = 5 }`

But once you define a type alias, such as:

```
type alias Model =
  { name : String
  , password : String
  , passwordAgain : String
  }
```

Elm automatically creates a constructor function for you that gives you *another way* to create a `Model`. This is a regular function it gives you, so it has to care about parameter order, and so it goes in the name the *direction of the properties in your type alias*. So here the type of `Model` is

```
Model : Name -&gt; Password -&gt; PasswordAgain -&gt; Model
```

U2GPAEU1L : (Where `Name` and everything else are `String`s, I just wanted to make it more clear than `String -&gt; String -&gt;...`
U5E99RPK6 : <@U2GPAEU1L> ah right, that makes sense. I was reading the `{}` part in the type alias as a record that was evaluated before being passed to `=`, but that's not record syntax (colon /= equals). so the whole type alias block is interpreted as one thing and the function is generated from it taking into account the order. thank you!
U2GPAEU1L : <@U5E99RPK6> Ya I get it, easy place to trip up (plus a lot of things have the same name which at first is confusing, but later you'll appreciate).
U3SJEDR96 : The positional encoding of magic constructor functions is pretty trippy :discoball:
U635MRFPY : hi, I am trying to build a single page app bootstrapped with `elm-community/elm-webpack-starter` but I am having issues due to the injected js from webpack
U635MRFPY : the problem is that the path to the main js is relative (src/static/main.js) and so if I e.g. have a nested route (like /accounts/recent) the js file can not be found
U635MRFPY : any tips on how to tell webpack to use `/src/static/main.js`?
U635MRFPY : got it, setting `output.public_path = '/'` seems to do the trick
U64FYS317 : Is there a way to apply a function to the result of a decoder? i.e. something like withDefault, turning `decoder (maybe (list string))` into `[]` if it was a `Nothing`?
U23SA861Y : so you want to use `map` and I believe what you want is `Maybe.withDefault` to be the function you map
U23SA861Y : `Json.Decoder.map (Maybe.withDefault []) otherdecoder`
U64FYS317 : thank you <@U23SA861Y>
U64FYS317 : That was rather simple :slightly_smiling_face:
U23SA861Y : it's supposed to be :slightly_smiling_face:
U2SR9DL7Q : Silly question but, can I send multiple commands in the same update function? On a related note, is it good practice to have updates trigger other updates?
U0LPMPL2U : in most cases, no

U0LPMPL2U : if you're trying do DRY your code, extract functions
U0LPMPL2U : `update` is for handling input from the outside world
U4872964V : <@U2SR9DL7Q> yes, Cmd.batch and no, it's not good practice, just do the work directly instead
U0LPMPL2U : for example:```
Foo -&gt; (setFieldOnModel model, Cmd.batch [cmd1, cmd2])
Bar -&gt; (setFieldOnModel model, Cmd.none)
```

U0LPMPL2U : the common model logic is extracted into the `setFieldOnModel` function and called from both branches of the `case`
U0LPMPL2U : the `Foo` branch uses `Cmd.batch` to trigger multiple side effects with commands
U2SR9DL7Q : hmmm... I had to use update so far because I needed random number generation. But now I have a sequence of operations and model changes that happen after that (for some game logic).
U2SR9DL7Q : But I think I get what you're trying to say.
U0LPMPL2U : can you post an example? It might be easier to give advice on a concrete scenario :slightly_smiling_face:
U0LPMPL2U : when you're generating several random numbers at once, it's often better to combine the generators into into a single one that returns a tuple or record and only use a single command
U2SR9DL7Q : Sure...```
SetGame newSet firstplayer -&gt;
            let
                players =
                    createPlayers newSet
            in
            ( { model | dominoes = newSet, game = Just &lt;| Game [] players firstplayer }, Cmd.none )

        ShuffleList -&gt;
            ( model, Random.generate SetFirstPlayer &lt;| shuffle model.dominoes )

        SetFirstPlayer newSet -&gt;
            ( model, Random.generate (SetGame newSet) (<http://Random.int|Random.int> 1 4) )
```
To explain, when the user clicks `Start Game`, `Shuffle List` is called and shuffles a set of domnioes. That's passed to `SetFirstPlayer` which also needs randomness to decide who goes first. That was passed to setGame to make the appropriate model updates.

U2SR9DL7Q : `SetGame` should be at the same level of indentation as `ShuffleList`.
U0LPMPL2U : Could all the randomness be done in one step? Shuffling dominoes and selecting the first player?