U3SJEDR96 : but indeed, `Program` is a type, not a function
U6G2ACUSX : Oh! It's like a constructor?
U3SJEDR96 : in this case, it's a type that holds functions for interfacing with your program, and those functions must have signatures that match one another in a specific way, but can work on any type of value otherwise
U4872964V : <@U6G2ACUSX> yes, it's like a type constructor
U6G2ACUSX : Wow. That was a lightbulb experience. Thank you so much!
U3SJEDR96 : I.e. `program` doesn't really care _what_ your `model` is, as long as it is consistent between `init`, `update`, `view` and `subscriptions` (which are the four functions you can pass to `Html.program`)
U611WQPL4 : My lightbulb is still kind of flickering. But it will be AWESOME when I get it. :slightly_smiling_face:
U3SJEDR96 : The type parameters are there to ensure you only use functions that make sense, given the types of things they get to work with. I.e. it doesn't make a whole lot of sense to calculate the greatest common divisor of a list of strings, the imaginary function `gcd` would work on a `List Int`, rather than a `List String` or a `List a`. On the other hand, calculating the number of entries in a list is independent of what type of data you're actually storing in them, so `List.length` works on `List a`. Having it _only_ work for a `List String` would be pretty annoying
U3SJEDR96 : as another example, a `Dict comparable value` allows making a dictionary where you can store an association between a `comparable` key and any type of `value`, as long as they're all the same type; so when you retrieve an element from a dictionary, you _know_ it will be of a certain type, it is guaranteed.
U37HUSJ4R : can anyone help me with something pretty simple? I have the following state:

```
type alias CallControls =
    { paused : Bool
    }


type alias Call =
    { number : String
    , controls : Maybe CallControls
    }
```

And I am trying to write an update function for paused? I can get it to work if it was `    , controls : CallControls` but struggling with the maybe

U37HUSJ4R : ```updatePaused: Bool -> Call -> Call
updatePaused newValue ({controls} as call) =
  { call | controls = { controls | paused = newValue } }
```

U37HUSJ4R : how can I wrap this in a `Just`?
U3SJEDR96 : `{ call | controls = Maybe.map (\controls -&gt; { controls | paused = newValue }) call.controls }`
U3SJEDR96 : unless you also want that to do something when `controls = Nothing`....
U3SJEDR96 : in which case you'd go `{ call | controls = Just { paused = newValue } }`
U3SJEDR96 : but then that's a little unrealistic :stuck_out_tongue:
U37HUSJ4R : brilliant thanks
U37HUSJ4R : I also think I might look into lenses
U37HUSJ4R : because I have quite a few nested props
U3SJEDR96 : eeeeh. I'd wait with that until you have a very firm grip on doing it without the
U0LPMPL2U : You probably don't need to nest as much as you think you do
U0LPMPL2U : Coming from OO, I tended to nest records _way_ too much
U37HUSJ4R : maybe not, I am probs going to end up going 2/3 levels deep
U37HUSJ4R : which isn't that many?
U3SJEDR96 : Heh, yeah, true. "encapsulate all the thing", I thought, before realizing that encapsulating records doesn't make much sense, since I can simply define a `func : { a | pauzed : Maybe Bool } -&gt; { a | pauzed : Maybe Bool }`. Encapsulating into types, on the other hand, helps in making impossible states impossible, whereas nesting records doesn't do much other than make them harder to work with
U37HUSJ4R : my thinking is certainly around impossible states
U37HUSJ4R : for example I don't want paused to be true if user isn't on a call
U3SJEDR96 : that imaginary `fun` above can _only_ touch `pauzed`, and is not aware of anything else in your record. So that basically creates the same guarantee as nesting it
```

U37HUSJ4R : so I have something like
U37HUSJ4R : ```type Status
    = Available
    | Wrap
    | OnCall Call
```

U3SJEDR96 : That might be extended to `Status = ... | OnCall Bool Call` though
U3SJEDR96 : which guarantees that you only have that bool if you're actually in a call
U3SJEDR96 : come to think of it `OnCall Bool Call` is basically `OnCall Call | Paused Call` anyway
U37HUSJ4R : true, but this is just a simple example
U37HUSJ4R : i also have transfer, hold etc
U6FFD2QG0 : Hi everyone, I'm running into something that seems like it should have a simple solution, but I can't figure out what that is. I need to construct an instance of `Cmd msg` as an alternative to `Cmd.none` in an if branch. I'm not using any outside effects or anything. Here's the relevant code snippet:```
update : Msg -&gt; Model -&gt; (Model, Cmd Msg)
update msg model =
  case msg of
    Tick newTime -&gt;
      let
          newTime = decTimer model
          newCmd = if newTime.activeTimer &gt; 0
                   then Cmd.none
                   else Cmd.Cmd TimerDone -- help!
      in
          (newTime, newCmd)
```

U0LPMPL2U : Is this to prevent duplication between the `Tick` and `TimerDone` branches of your `update` ?
U6FFD2QG0 : yeah, basically
U3SJEDR96 : I would suggest taking the contents of your `TimeDone` branch, putting it into a separate function, and replace that with```
TimerDone ->
    timerDone model
Tick newTime ->
    if .activeTime (decTimer model) > 0 then (newTime, Cmd.none) else timerDone model
````
````