U6FECHN3B : that's a good point too
U3SJEDR96 : I've had that, too; though in my case it was partially due to sharing config between multiple machines using `mackup` + dropbox. Or at least, it stopped happening _constantly_ after turning `mackup` off
U3SJEDR96 : There's little point to failing your build because a compiled artefact makes different stylistic choices than your team does - which is what semicolons are. It's not really an ES5 vs ES6 thing, even. Though I don't see Elm generating ES6 for quite some time, as we tend to aim for compatibility, to the point of providing a polyfill for requestAnimationFrame
U3SJEDR96 : So yeah, that would be my recommendation. If that's not possible for whatever reason, you could look into post-processing
U6FECHN3B : it won't fail a build fortunately just ugly in our tooling
U6FECHN3B : semi-new to the Elm world so figured I'd ask
U3SJEDR96 : Fair enough! :slightly_smiling_face:
U3SJEDR96 : But yeah, I'd exclude the compiled artefacts from linting and VCS, generally
U6FECHN3B : excluded indeed
U6D3ERLA1 : What do I do if I end up with nested `Maybe`s ex from doing chained list operations
U6D3ERLA1 : ```   case (List.drop (List.length xs - 2) xs) of -- Maybe.Maybe
        Just val -&gt;
            Just val
        Nothing -&gt;
            Nothing
```

U3SJEDR96 : `List.drop` doesn't return a `Maybe`, tho? Some more context might help clarify things :slightly_smiling_face:
U3SJEDR96 : nested maybe's are _generally_ something you avoid using `Maybe.andThen`, though if you do end up with a nested `Maybe`; you can `Maybe.andThen identity` to unwrap it a single level
U0LPMPL2U : :thumbsup: to using `Maybe.andThen` to chain operations that return Maybe
U6D3ERLA1 : Ahh that's how to get the values out of the `Maybe`
U3SJEDR96 : <@U6D3ERLA1> `andThen identity` only works on nested maybe's, though - a single level of maybe is either pattern matching or using `Maybe.withDefault (defaultValue)`
U611WQPL4 : I've read (and reread) a lot of the Intro to Elm documentation.  But I keep struggling with one part.  You can see it on the MVCTodo example.  <https://github.com/evancz/elm-todomvc/blob/master/Todo.elm#L26-L33>.  What does `main : Program (Maybe Model) Model Msg` mean?   I can follow the syntax when there are `-&gt;` - i.e. `updateWithStorage : Msg -&gt; Model -&gt; ( Model, Cmd Msg )` but not that type.
U611WQPL4 : ^^ You see it with lists too.  `List Int`, `List a` and `Cmd msg`.  Is this a compound type?
U3SJEDR96 : it means you're dealing with a type that is _parametrized_. In case of `List`, it means it holds a type of value, which doesn't matter to how the list is actually implemented, but _does_ matter when actually using such a `List`
U611WQPL4 : Yeah, you see that in C++ with generics/templates.  - i.e. `List&lt;Int&gt;`.
U3SJEDR96 : in terms of `List`, it is essentially a union type `type List a = Empty | Cons a (List a)`, meaning it's either empty, or holds a value of type `a` and then a nested list that also contains either nothing a value of type `a` and some mroe values etc etc
U3SJEDR96 : `Maybe` is a little simpler: it's literally `type Maybe a = Nothing | Just a`.
U3SJEDR96 : Yeah, it's pretty similar to generics, indeed
U611WQPL4 : Thanks <@U3SJEDR96>.  Disclaimer:  I've asked this question before.  :slightly_smiling_face:
U3SJEDR96 : now, `Program` is a little special, as it is an opaque type and we don't know what it actually looks like. However, you can pass it a number of functions; and the types of those fucntions are also variable. A `Program flags model msg` is a `Program` that sends `flags` to its `init` function, which will return a `model` and a `Cmd msg`, has an `update : msg -&gt; model -&gt; (model, Cmd msg)`, and so on, all with *consistent* types for all those type parameters, throughout
U3SJEDR96 : This allows creating a compile-time guarantee that you cannot possible have an initial model that is simply an `Int`, together with an `update` function that only works on `String`s
U3SJEDR96 : I do remember the "leap" from `List a` to `Program flags model msg` to be a significant one, though, so it's okay if you _don't quite get it just yet_!
U6G2ACUSX : How do you read that then? 'List String' is a list of strings. So then 'Program flags model msg' is a Program of flags of model of msg.. or?
U6G2ACUSX : (I also found this hard to grasp when reading the intro)
U4872964V : <@U6G2ACUSX> it's just the types for flags, model and messages that your program uses that go into those places
U4872964V : So Elm knows what types your program uses
U6G2ACUSX : Ok, I guess html.program is a special case, so I don't really need to understand it. Especially since I'm
```

completely new to Elm and functional programming in general. But how would you create a function that has that sort of signature? Can you even?
U6G2ACUSX : Oooor, it's not a function at all? It's a type?
U4872964V : <@U6G2ACUSX> when you define types, you can add type parameters to them, yes
U4872964V : It's like you can have small "holes" in your type that is filled in with the parameters
U3SJEDR96 : Sure - `threeTuple : a -&gt; b -&gt; c -&gt; (a, b, c)`. Or `flags`, `model` and `msg` :slightly_smiling_face:
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 -&gt; Call -&gt; 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
```