U3SARGL7Q : <@U4872964V> I am trying to conditionally parse JSON, I have JSON's delivered over WS such as `{"event" : "abc", params: {}}`, `{"event" : "def", params: {"someparam" :123}}`
U3SARGL7Q : I want to write JSON decoder that outputs the Msg.Abc or Msg.Def, where parameters to these messages are different depending on the message that arrived
U2M4VPZ9D : <@U3SARGL7Q> Can you maybe use andThen as in <http://package.elm-lang.org/packages/elm-lang/core/5.1.1/Json-Decode#andThen> ?
U2M4VPZ9D : Depending on what the event string is set to, you chain it the right decoder.
U3SARGL7Q : <@U2M4VPZ9D> I already reached the point where I know that it is about andThen but I can't proceed
U3SARGL7Q : here's my code
U3SARGL7Q : ```module Subscriptions exposing
  (subscriptions
)

import WebSocket
import Json.Decode as Decode

import Models exposing (Model)
import Msgs exposing (Msg)

{-| Listens on messages on the WebSocket channels to the backend.
-}
subscriptions : Model -> Sub Msg
subscriptions model =
  WebSocket.listen "<ws://localhost:5000/api/player/v1.0/stream>" playerMessageHandler

{-| Handles incoming messages that can be sent by the player.
-}
playerMessageHandler : String -> Msg
playerMessageHandler payload =
  case Decode.decodeString playerMessageDecoder payload of
    Err err ->
      Msgs.OnPlayerUnexpectedMessage err

    Ok msg ->
      msg

type alias PlayerEvent =
    { event : String
    , tracks: List }

{-| Decoder for incoming messages that can be sent by the player that is
converting JSON into message appropriate for the payload.
-}
playerMessageDecoder : Decode.Decoder Msg
playerMessageDecoder =
  Decode.field "event" Decode.string
  |> Decode.andThen interpretPlayerEvent

interpretPlayerEvent : String -> Decode.Decoder Msg
interpretPlayerEvent event =

```
  case event of
    "snapshot" -&gt;
      Decode.succeed Msgs.OnPlayerSnapshot

    "track-position" -&gt;
      Decode.succeed Msgs.OnPlayerTrackPosition

    _ -&gt;
      Decode.fail event
```

U3SARGL7Q : but the thing is that the remaining JSON fields depend on what is in the "event" field and I struggle to find a way to do this
U2M4VPZ9D : <@U3SARGL7Q> What does your snapshot structure look like? Did you write a decoder for that?
U3SARGL7Q : there are two types of messages
U3SARGL7Q :
```{"event":"snapshot","tracks":[{"state":"playing","position":82940,"track":{"rotation_id":"whatever2","kind":"auto","id":793,"file":{"na
Hope.mp3","id":"dd8f2ff7-1ee0-45c5-b8d8-5be625acccb7","duration":93037},"fade_out_at":"2017-07-24T19:25:18.767898+02:00'
```

or

```
{"event":"track-position","track":{"rotation_id":"whatever2","kind":"auto","id":793,"file":{"name":"32
Hope.mp3","id":"dd8f2ff7-1ee0-45c5-b8d8-5be625acccb7","duration":93037},"fade_out_at":"2017-07-24T19:25:18.767898+02:00'
```

U3SARGL7Q : I know how to write decoder if the schema is always the same
U3SJEDR96 : `field "event" string |> andThen (\eventString -> case eventString of "snapshot" -> snapshotDecoder [...])`
U3SJEDR96 : oh, you already have that
U3SJEDR96 : well, sort of
U3SJEDR96 : currently you're just succeeding with a message, rather than decoding the data
U2M4VPZ9D : Yes, I think you just need to write the decoder for the snapshot tracks and the decoder for the track-position track.
U3SJEDR96 : but you could do `Json.map Msgs.OnPlayerSnapshot playerSnapshotDecoder` instead
U3SJEDR96 : exactly
U2M4VPZ9D : Sorry sorting out kids bath at the same time :slightly_smiling_face:
U3SARGL7Q : But my exact problem is how to instruct elm to pick up the right decoder depending on the value of "event" field?
U3SARGL7Q : andThen in <@U3SJEDR96> example does not have access to another top-level fields from the JSON
U3SJEDR96 : it does, depending on if you did `field "foo" (string |> andThen ..)` or `field "foo" string |> andThen ..` - the first runs within `field "foo"`, the other runs at the same level as `field "foo"`. You can't go "up", but you can definitely stay in that same context
U3SJEDR96 : <https://ellie-app.com/3Qm3MMZJFgba1/0> for a _very_ q'n'd example
U3SJEDR96 : now you'd "simply" need to write proper decoders for snapshot and trackPosition, and you're all set
U3SARGL7Q : hmm i'll check it now
U3SJEDR96 : Your solution is like 90% there, hence my being confused :slightly_smiling_face:
U3SARGL7Q : hmm seems to work
U3SARGL7Q : why that parentheses matter?
U3SARGL7Q : in that particular case?
U3SARGL7Q : guys i am programming for years but elm syntax is not readable at all at the beginning
U0LPMPL2U : <@U3SARGL7Q> In Elm, arguments are separated by spaces, not commas. Thus if you are nesting functions, you need to wrap the sub-functions in parens
U4WH8STNX : <@U3SARGL7Q> it is just a matter of getting used to it.
U0LPMPL2U : unlike C-style languages, parens go around both the function name _and_ the function arguments
U0LPMPL2U : so `(sum 1 2)`, not `sum(1, 2)`
U3SARGL7Q : anyway you're awesome, thanks for help!
U3SJEDR96 : You were really 90% there, just needed a nudge :slightly_smiling_face:
U3SARGL7Q : Ok one more question: how to detect that websocket connection is down? :slightly_smiling_face: seems

that it is hidden and there's no way to access it easily without using WebSocket.LowLevel?
U3SJEDR96 : Yeah, it's a design decision that was made in that library that you can't actually know it. It implements incremental backoff and builds up a queue of outgoing messages, but you can't actually know whether you're connected. It's a little controversial, I think there are some github issues about it
U2SR9DL7Q : I have a maybe odd question. I have a type made from four of the same type, like `type FunType = Other Other Other Other` I'm trying to create a map2 essentially for `FunType` where the function would look like `map2 : (Other -> a -> Other) -> FunType -> List a -> Funtype` ... it should essentially take the first four from any list, with a default if the list doesn't have at least four things.
U3SARGL7Q : <@U3SJEDR96> I need to inform the user if the connection went down, so indeed I will consider this controversial.
U0LPMPL2U : <@U2SR9DL7Q> you can't use `Other` in a type signature. `Other` is a value / constructor, not a type :slightly_smiling_face:
U3SJEDR96 : It seems to be a type, too
U0LPMPL2U : oh I see :smile:
U3SJEDR96 : though that type only has 3 `Other` values, and 1 serving as the tag :stuck_out_tongue:
U3SJEDR96 : so you can do `case list of a :: b :: c :: d :: _` to see if there are 4 entries _and_ get them out at the same time
U2SR9DL7Q : sorry... I wanted unambiguous names for the example but it didn't work. In production, I have a `type Players = FourPlayers Player Player Player Player`
U3SJEDR96 : which I suppose is what you're actually asking :slightly_smiling_face:
U3SJEDR96 : That's still only 3 players, though :smile:
U3SJEDR96 : The first one is the tag of your type, the next 3 are the values it holds
U3SJEDR96 : like `type MyType = String String` can only hold a single `String`
U2SR9DL7Q : <@U3SJEDR96> fixed it. it's really four. and that case statement solves the issue of more than four but not less than four in the list
U3SJEDR96 : well less than four is `_ ->` :wink:
U3SJEDR96 : or `a :: b :: c :: _` and so on if you want to be explicit about it. Or even `[ a, b, c]` so the order doesn't matter as much
U3SJEDR96 : (the order of your case-statements, that is)
U2SR9DL7Q : Hmmm... so when you say `case list of a :: b :: c :: _`, elm interprets that as _"name as many values in this list as you can up to the first three values?"_
U3SJEDR96 : ```case yourList of
  a :: b :: c :: d :: _ -> "I have 4 or more values!"
  a :: b :: c :: _ -> "Exactly 3, because 4 or more is already matched"
  val1 :: val2 :: _ -> "other names work, too"
  _ -> "1 or 0, and I'm lazy to type them out"
```

U2SR9DL7Q : Ahhh... okay, no spooky magic, just regular patterns matching.
U3SJEDR96 : So what's important here is that you can think of `[ 1, 2, 3 ]` as sugar for `1 :: 2 :: 3 :: []`
U3SJEDR96 : and then the whole thing becomes regular pattern matching :slightly_smiling_face:
U2SR9DL7Q : That... can work. Permit me to expand scope one step further. I have this recurring problem with lists, where I keep dealing with handling edge cases in lists that don't exist. In this case, when constructing the players, each `Player` has an order, represented by an integer. I wanted a neat way to construct the players, some thing like `List.map (\x = Player x ) &lt;| List.range 1 4` ... and this will work, but I'll need to do the pattern match as you suggest, which is _fine_.
U2SR9DL7Q : But I feel strange handling cases that will never happen, and it makes me feel like I'm not writing this properly. There should be a way to structure the code so that isn't necessary is my intuition.
U2SR9DL7Q : If my types make the compiler ask for cases that are impossible, I must have the wrong mental model of my types...
U3SJEDR96 : A list has an arbitrary number of entries, and it sounds like you need a type that has a stricter number; for example your `Players` type, or perhaps a record of 4-tuple..
U3SJEDR96 : You'll end up with somewhat less "neat" code, in the sense that it will be more explicit about what it does and how it works, but _neat_ comes at a cost, too
U2SR9DL7Q : the `Players` type became a thing because List of Players was peppering maybes everywhere
```