U23SA861Y : to make a list encoder `participantListEncoder pList = List.map participantEncoder pList |&gt; JE.list`
U23SA861Y : if I want to transform a List of some type into a list of another type you use `map`
U23SA861Y : In this case you would have a list of participants and you want a list of values
U23SA861Y : so you call `List.map` with a function that transforms a single participant into a `Json.Encode.Value`
U23SA861Y : you can then pass that list of values to `Json.Encode.list` to construct a new value
U6D3ERLA1 : :bulb:
U6D3ERLA1 : that makes sense
U6D3ERLA1 : pipe it back to values
U6D3ERLA1 : <https://gist.github.com/254b044a54ec101c5ed7392eb8df5677>
U6D3ERLA1 : <@U0CQ254F5> <@U23SA861Y> Thanks for data pipeline 101 :train:
U1ZFF0E5P : Hi, I'm trying to make the following decoder polymorphic instead of hard-coded to String for result, but I can't figure out how to do it```type alias DictItem =
    { result : String
    , entities : Entities
    }

dictItemDecoder : Decoder DictItem
dictItemDecoder =
    decode DictItem
        |> required "result" string
        |> required "entities" entitiesDecoder```

U1ZFF0E5P : so I changed the alias to this, but I'm stuck at the decoder level: ```type alias DictItem a =   { result : a
    , entities : Entities
    }```

U1ZFF0E5P : I tried this but it obviously doesnt work ```dictItemDecoder : Decoder (DictItem a)dictItemDecoder =
    decode (DictItem a)
        |> required "result" a
        |> required "entities" entitiesDecoder```

U153UK3FA : <@U1ZFF0E5P> how do you expect the compiler to infer the type of `a`?
U1ZFF0E5P : that is a very good question
U1ZFF0E5P : essentially I'm moving all my ids from strings to wrapped types like so: ```type StudentCourseClassId   = StudentCourseClassId String```

U1ZFF0E5P : but yeah thinking about it there is no way for the compiler to guess that
U1ZFF0E5P : so either I "duplicate" this decoder for each id type, or I can probably put them in a union type and use this
U23SA861Y : ok so one thing about elm is that it's not polymorphic, if you want to make something type agnostic you will have to have a parameter which passes down how to deal with that type.
U1ZFF0E5P : yeah, I'm still struggling to think this way (coming from dynamic languages)
U1ZFF0E5P : is this where typeclasses would be useful? as in I can pass down "a" and define on it's typeclass level how to be decoded?