U0LPMPL2U : which I guess could make sense if you're transforming players
U0LPMPL2U : The problem is that the `map` and `map2` would only allow functions of type `Player -&gt;Player` and `Player -&gt; Player -&gt; Player` respectively
U2SR9DL7Q : Yeah, what I want isn't really map. It's a map-player humunculus monstrosity
U0LPMPL2U : Perhaps you do want a parameterized type?
U2SR9DL7Q : But really just a way to take 4 values from a list, andapply them to each player in players, which i think <@U3SJEDR96> solution would do.
U0LPMPL2U : `type FourItems a = FourItems a a a a` ?
U0LPMPL2U : Then it would make sense to have a function `FourItems.map2`
U3SJEDR96 : Perhaps it makes sense to be explicit about the arguments - `mapPlayer1and2 : (a -&gt; Player -&gt; Player) -&gt; a -&gt; a -&gt; Players -&gt; Players` something like that?
U3SJEDR96 : I.e. 2 `a`'s for 2 players?
U2SR9DL7Q : <@U0LPMPL2U> That's more abstraction than I may need. It's always 4 players. Players functions like a fixed list of 4 things. Because it's not a list, I have to explicitly handle mapping functions that interact with it. I was hoping that I was missing a clever way to let it interact with lists. So if I have a list of four items, I could use it in a function with players and get a new Players back. Thats why it feels like map two functionality in my head.
U2SR9DL7Q : I have this function```
```
playerMap : (Player -&gt; Player) -&gt; Players -&gt; Players
playerMap func players =
    let
        (FourPlayers p1 p2 p3 p4) =
            players
    in
    FourPlayers (func p1) (func p2) (func p3) (func p4)
```

U0LPMPL2U : yup, that makes sense
U3SJEDR96 : (you can actually do that pattern in the declaration)
U3SJEDR96 : i.e. `playerMap func (FourPlayers p1 p2 p3 p4) = `...
U2SR9DL7Q : which works just fine for transforming players. I just wanted to find the best solution for doing the same thing but with a list added `playerMap2 : (Player -&gt; a -&gt; Player) -&gt; Players  -&gt; List a -&gt; Players`

U2SR9DL7Q : so i could update all four players at once with a list of four values
U2SR9DL7Q : So it's not a true map, because it's not two lists or two players. Its players and a list.
U3SJEDR96 : but why not `a -&gt; a -&gt; a -&gt; a` or rather than a `List`? When you're calling it, I'm assuming you're not dealing with a list, but rather creating one for the purpose of sending it through?
U2SR9DL7Q : I should also admit that the purpose for this isn't to get something to work (the logic works). But it's all very crude and clunky, and I'm trying to learn and use more FP design in function construction.
U2SR9DL7Q : <@U3SJEDR96> that... could work if I'm constructing the list for this purpose :slightly_smiling_face: (which I almost always am)
U2SR9DL7Q : This game app unfortunately has a lot of stuff passed around, and a lot of maybes and before I move on to the next step I want to refactor as best I can. All the examples I see have really simple functions with neat piping and clever usage of the core library and I'm _want_ that as well. I know it'll take time and effort though, before I can think in those sort've efficient chains of functions.
U2SR9DL7Q : <@U3SJEDR96> <@U0LPMPL2U> thanks!```
```
playerMap2 : (Player -&gt; a -&gt; Player) -&gt; Players -&gt; ( a, a, a, a ) -&gt; Players
playerMap2 func (FourPlayers p1 p2 p3 p4) ( v1, v2, v3, v4 ) =
    FourPlayers (func p1 v1) (func p2 v2) (func p3 v3) (func p4 v4)
```

U0LPMPL2U : nice use of a tuple there :thumbsup:
U6D3ERLA1 : teehee can I ask some noobish questions here?
U6D3ERLA1 : If I do:`l2 = List.range 1 19999`
I can do:
`l3 = List.filter (\n -&gt; n % 99 == 0) l2`
but not:
`l3 = l2.filter ...`

U6D3ERLA1 : Is there a syntactic equivalent to the dot syntax in js?
U1UGYHGCA : Nope

U681TBBUP : No there is not; having functions attached to data is an object oriented concept
U0LPMPL2U : <https://elmlang.slack.com/archives/C192T0Q1E/p1500924219522502>
U0LPMPL2U : All beginner questions welcome here :slightly_smiling_face:
U0LPMPL2U : You can get close to OO dot notation with the forward pipe operator