U48AEBJQ3 : <@U2SR9DL7Q> Are you just trying to write `List.map2 (List.map &lt;&lt; (,)) xs (tails xs) |&gt; List.concat`?
U2SR9DL7Q : Okay, so we're almost there. I have```
createDominoes : Int -&gt; List Domino
createDominoes highest =
    List.map (\x -&gt; List.map (\y -&gt; Domino x y &lt;| List.range x highest)) &lt;| List.range 0 highest
```
The `Domino` here is just a type defined by two integers.

But I've messed it up in the inner function so the compiler is yelling
```
a -&gt; Domino

But the left argument is:

Domino
```

U2SR9DL7Q : I just added a new comment in beginner channel that adds more clarity. But you're speaking in haskellian right now, and I haven't done that in _awhile_. It's honestly going to take me a few minutes to parse that statement and then I can tell you.
U3SJEDR96 : I think you wnt to move that `)`: `List.map (\x -&gt; List.map (\y -&gt; Domino x y) &lt;| List.range x highest ) &lt;| List.range 0 highest`
U48AEBJQ3 : <https://ellie-app.com/3KQ5Rq7VdHNa1/0> ?
U2SR9DL7Q : That... worked? It says I've created a list of a list of dominoes. But If I can just flatten that, I should be fine.
U3SJEDR96 : `concatMap` to the rescue
U2SR9DL7Q : <@U48AEBJQ3> your solution is probably the more clever, FP way to do it, but I'll have to sit and study it.
U3SJEDR96 : or what <@U48AEBJQ3> did, which is nice :slightly_smiling_face:
U3SJEDR96 : the observation that for every element in the range, you only want to make combinations of the element and everything that follows is clever :slightly_smiling_face:
U2SR9DL7Q : Yes, but it's very imperative thinking. I've just made the elmy equivalent of ```
for i in range(0, num):
    for j in range(i, num):
```

U3SJEDR96 : yeah, and <@U48AEBJQ3> uses the same _idea_ in their implementation. I was actually remarking on his code, even though you'd done the same thing (but I hadn't realized it because I was trying to spot the bad code, rather than understand it)
U48AEBJQ3 : I guess the function-fu of `List.map &lt;&lt; (,)` is probably a bit much for learning. You can read it as:```
\x ys -&gt;
    List.map (\y -&gt; (x, y)) ys
```

U57KYFW67 : ```[1,2,3,4,5] |&gt; andThen (\x -&gt; [1,2,3,4,5] |&gt; andThen (\y -&gt; if x &lt; y then [(x, y)] else []))
[(1,2),(1,3),(1,4),(1,5),(2,3),(2,4),(2,5),(3,4),(3,5),(4,5)]
    : List ( number, number )
```

U57KYFW67 : (how do I do code blocks in Slack??)
U48AEBJQ3 : triple backtick on its own line before and after the block
U57KYFW67 : tyty
U2SR9DL7Q : <@U48AEBJQ3> that makes it all much clearer. Unfortunately I never did enough haskell to get comfortable with all the inline functions. I'll remember this one now though.
U57KYFW67 : That code I posted does what the OP wanted. There's only two tricks to know: `andThen` allows you do iterate in a way a bit analogous to a for loop and the `if x &lt; y` condition will either append `(x,y)` or else it will append nothing.
U57KYFW67 : (to be reallllly handwavy)
U2SR9DL7Q : <@U57KYFW67> you got it to work! that's what I tried initially, but the exact nature of what `andThen` is

(binding operation for Lists that are monad typeclasses) makes me wary of using it too much.
U57KYFW67 : `andThen` is pretty neat, but the name doesn't make much sense.