U69D8R59S : Wow. You all are fast and awesome. Thank you!
U2D07QZN3 : Http.getString returns HttpRequest String  How do I take that apart to get the string?
U4872964V : you use `Http.send` to create a command that you return from your update function
U62UFEG4D : Hello happy Elm people, hope you are all doing amazing today :slightly_smiling_face: !I am trying to apply a function to a list of tuples.
I intuition I need to unpack (_uncurry?_) the tuples, but so far I have miserably failed.
Here is an example to give an idea:

```
points =
   (List.map2
      (,)
      (arrayX |&gt; Array.toList)
      (arrayY |&gt; Array.toList)
   )
      |&gt; List.map Geometry.Point
```
any help appreciated ! Thanks,

U4872964V : <@U2D07QZN3> <https://guide.elm-lang.org/architecture/effects/http.html>
U4872964V : <@U62UFEG4D> well, you uncurry the function to make it work on tuples, so```
List.map (uncurry Geometry.point)  yourListOfTuples
```
if I understand your request

U4872964V : but from your description it sounds like you have to Arrays that you want to apply the function to
U62UFEG4D : Awesome, it works <@U4872964V>!
U62UFEG4D : I did not get that uncurry applied to a function... not to arguments
U62UFEG4D : thanks!
U4872964V : <@U62UFEG4D> you don't have to make intermediate tuples though, you should be able to apply your function directly```
points =
   List.map2 Geometry.Point
      (arrayX |&gt; Array.toList)
      (arrayY |&gt; Array.toList)
```

U62UFEG4D : oh wow!
U62UFEG4D : makes totally sense, thank a lot <@U4872964V>!
U4872964V : but now you got to learn uncurry so it's all good :slightly_smiling_face:
U62UFEG4D : exactly hehe!
U0FP80EKB : Don't need to pipeline (personal style, though)```
points =
   List.map2 Geometry.Point
      (Array.toList arrayX)
      (Array.toList arrayY)
```

U4872964V : yes, that's personal style, I sort of like the "actual" argument being first so that it's visible
U0FP80EKB : Definitely
U5AEH3L05 : Okay, this seems like this should be straightforward: `event.target.value` gives me a float value encoded as a string, so `value : "100"`. How do I decode that into a float?
U5AEH3L05 : If I do that as `Json.Decode.float`, it chokes because of the quotation marks
U5AEH3L05 : My current best approach is ```    on "input" <|        Json.map tagger <|
         Json.andThen
            (\value ->
               case String.toFloat value of
                  Ok float ->
                     Json.succeed float
```

```
                Err err -&gt;
                    Json.fail err
            )
            targetValue
```

U236M9FH9 : Decode as a string & use `String.toFloat` with `andThen`, with `fail` as the error case & `succeed` with the success case
U5AEH3L05 : Which seems wildly verbose
U5AEH3L05 : Haha, good timing lysergia :slightly_smiling_face:
U236M9FH9 : You can use `fromResult` from `json-extra`: <http://package.elm-lang.org/packages/elm-community/json-extra/2.3.0/Json-Decode-Extra#fromResult>