U663M2MB7 : Has anyone used <https://github.com/rtfeldman/elm-css> without having it create an external css file? I would like to use it in my code, and have it style my elements when running `elm-reactor` but I am confused as to whether this is an possible.
U46JV6X3K : What's the best way to encode a record to a JS object?
U68RXKS5D : Check out Json.Encode
U68RXKS5D : <http://package.elm-lang.org/packages/elm-lang/core/5.1.1/Json-Encode>
U0CL0AS3V : <@U663M2MB7> definitely doable - ask in <#C0HJVT881|elm-css> and folks can help!
U48AEBJQ3 : <@U46JV6X3K> Presuming you want to make a JS object:```
type alias MyRecord =
  { foo : Int
  , bar : String
  }

encodeMyRecord : MyRecord -> Encode.Value
encodeMyRecord myRecord =
  Encode.object
    [("foo", <http://Encode.int|Encode.int> myRecord.foo)
    , ("bar", Encode.string myRecord.bar)
    ]
```

U663M2MB7 : <@U0CL0AS3V> thanks man, will do! :slightly_smiling_face:
U65B9414J : Folks posted my first blog on Elm. This was an issue for me and my team members while understanding Elm. It's very basic stuff. <https://blog.bigbinary.com/2017/07/12/difference-between-type-and-type-alias-in-elm.html> . If anyone has any feedback then let me know. Thanks everyone and keep on Elming.
U3LUC6SNS : I need to frequently to the below after starting `elm-repl`.  Is there a way of putting this in a script or something to avoid repetitious typing?
```
> import LatexParser.Latex exposing(..)
> import LatexParser.Parser exposing(..)
> import Parser exposing(run)
```

U494Y62N7 : huh, i guess it would be nice to do something like `elm-repl YourCode.elm`
U3LUC6SNS : <@U494Y62N7> I tried this:```
jxxmbp:koko_client carlson$ elm-repl start.elm
Unhandled argument, none expected: start.elm
```

U494Y62N7 : Yea, I was thinking that would be nice. F# has the ability where you can just highlight code & send it to the repl.
U494Y62N7 : Clojure does as well.
U494Y62N7 : I guess it's time to start learning Haskell :slightly_smiling_face:
U3LUC6SNS : Another question. In the `elm-repl` I do the following:```
 r = run latexList "a b c\nx y z\n"
Ok { value = [Words { value = ["a","b","c"] },Words { value = ["x","y","z"] }] }
  : Result.Result Parser.Error LatexParser.Parser.LatexList
```
Is there an easy way to extract the part after the `Ok`?  Working with multi-line code in the repl is a pain.

U494Y62N7 : The only thing I can think of is to write a function with `Result.withDefault` and then have the `value` just be an empty array if it fails
U494Y62N7 : Sorry :disappointed:
U153UK3FA : <@U3LUC6SNS> `Result.withDefault` is what you want. But you might just want `Result.map` if you're planning on using that value for something
U2GPAEU1L : If you want a tutorial on it:
<http://codetidbit.com/#view/story/58f7ac012bdce7111285c2ea>

These 5 mini-tutorials take you through everything you need to know about encoding and decoding in Elm
```

U2U94G0QG : <@U3SJEDR96> My attempts at `at : List String -> Decoder a -> Decoder a` :
U2U94G0QG : ``` -- Haha, what a fool I am... But it works!
```
at : List String -&gt; Decoder a -&gt; Decoder a
at location decoder =
    let
        reorder =
            List.reverse location
    in
        case reorder of
            thing :: [] -&gt;
                field thing decoder

            thing :: moreThings -&gt;
                at (List.reverse moreThings) decoder
                    |&gt; field thing

            _ -&gt;
                fail "I don't understand that location."


 -- That's better... Say! This looks like a fold...
at : List String -&gt; Decoder a -&gt; Decoder a
at location decoder =
    case location of
        thing :: [] -&gt;
            field thing decoder

        thing :: moreThings -&gt;
            field thing decoder
                |&gt; at moreThings

        _ -&gt;
            fail "I don't understand that location."


 -- So pretty... This is how it's done in Json.Decode
at : List String -&gt; Decoder a -&gt; Decoder a
at location decoder =
    List.foldr field decoder location
```

U5P1BDUM8 : Is there any way to hide some sort of warnings on `elm-make --warn foobar.elm` like `fno-warn-*` pragmas of Haskell?I do not want to show following warnings on specific files.

```
Top-level value `foobar` does not have a type annotation.
```

U2U94G0QG : leave off the `--warn` part
U5P1BDUM8 : I want to show other sorts of warnings...