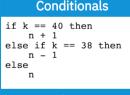


	Litovolo	Linto	Аннессия	Custom Types	Type Appetations	Dectmoduring
elm	Literals True/False: Bool 42: number (Int or Float) 3.14: Float 'a': Char "abc": String """multi-line string"""	Lists A collection of items of the same type [1,2,3,4] 1 :: [2,3,4] 1 :: 2 :: 3 :: 4 :: [] Records	Arrays Array.empty Array.fromList Array.toList Array.get Array.set Dictionaries	Custom Types Custom Types start with an upper case letter type User = Regular String Int Visitor String	Type Annotations answer : Int answer = 42 factorial : Int -> Int factorial n = List.product (List.range 1 n)	<pre>Destructuring sum addends = let (a, b) = addends in a + b sum (a, b) = a + b</pre>
Comments a single line comment {- a multi-line comment {- can be nested -} -} Trick to comment blocks of code {} add x y = x + y}	Tuples Can contain 2 or 3 items of different type. (1, "2", True) The Elm Architecture Browser.sandbox Browser.element Browser.document Browser.application headless Platform.worker	A collection of key/value pairs, similar to objects in JavaScript point = { x = 0, y = 0 } point.x == 0 List.map .x [point, point2 { point x = 6 } { point x = point.x + 1 } , y = point.y + 1 } Extensible Records have at least certain fields: f : { b key : a } -> a f = .key	Dict.empty Dict.fromList Dict.toList Dict.get Dict.update	Type Aliases Type Aliases start with an upper case letter type alias Name = String type alias Age = Int info : (Name, Age) info = ("Steve", 28) type alias Point = {x: Float, y: Float} origin : Point origin : {x = 0, y = 0}	<pre>distance : {x : Float, y : Float} -> Float</pre>	<pre>f list = case list of []</pre>
Functions Functions start with a lower cas letter. No parentheses or commarguments or code blocks. square n = n^2 hypotenuse a b = sqrt (square a + square a)	Anonymous functions with "\", that resemble lambda "\" square = \n -> n' squares = List.map (\n ->	Html.lazy Html.keyed Debugging n^2 Debug.toString Pebug log	Route = Blog Int Parser = oneOf ap Blog (s "b ap User (s "u	Routing osing (s,(),int,string, User String Comment S log"int) ser"string) ser"stringser"string </td <td>Advanced Types oneOf,map) Opaque types don't expose constructors. Phantom type: type Currency a</td> <td>number (Int, Float) appendable (String, List a) comparable (Int, Float, Char, String,</td>	Advanced Types oneOf,map) Opaque types don't expose constructors. Phantom type: type Currency a	number (Int, Float) appendable (String, List a) comparable (Int, Float, Char, String,



n		From JS, start Elm with flags and talk to these ports:			
Commands REPL		<pre><div id="app"></div> <script src="elm.js"></script> <script></pre></td></tr><tr><td>elm repl elm init elm reactor elm make elm install elm bump elm diff</td><td>:exit :help :reset Backslash() for multi-line expressions</td><td><pre>var app = Elm.Main.init({ node: document.getElementById('app'), flags: { key: 'value' } }); app.ports.prices.send(42); app.ports.time.subscribe(callback); </script></pre>			
elm publish		_, _			



elm-format elm-test elm-doc elm-doc-preview elm-spa elm-live/elm-go elm-json elm-review elm-graphql

```
Operators
  _ * /
                    math
                    int division
                    equality
< > <= >= max min comparison
not && || xor
                    booleans
                     append
```

fancy math

functions

bitwise

cons

modBy remainderBy and or xor <| |> << >> cument.getElementById('app'),

==

Most can be used in "prefix notation" too: a + b == (+) a b

Modules Imports

import List -- preferred import List as L import List exposing (..) import List exposing (map, foldl) case xs of import Maybe exposing (Maybe) import Maybe exposing (Maybe(..)) Side Effects Task/Cmd

Task.perform Task.attempt Task.andThen Cmd.batch

Available at ellie-app.com

update = update

import Browser

module Main exposing (main)

Hello World

div [] [text "Hello World!"]

Hello World with Elm-UI

Pattern Matching

module Main exposing (main)

import Element exposing (...)

module Main exposing (main)

import Html exposing (..)

main =

main =

layout [] <

case maybeList of

[] ->

0 -> 1

1 -> 1

case n of

Just xs -> xs

Nothing -> []

Nothing

first :: rest ->

Just (first, rest)

_ -> fib (n-1) + fib (n-2)

```
-1
                             import Html exposing (..)
                             import Html.Events exposing (..)
                             type alias Model = { count : Int }
                             initialModel
                                           = { count = 0 }
                             type Msg = Increment | Decrement
el [] [text "Hello World!"]
                             update msg model =
                                case msg of
                                   Increment ->
                                   Decrement ->
```

Counter

```
{ model | count = model.count + 1 }
        { model | count = model.count - 1 }
view model = div []
 [ button [onClick Increment] [text "+1"]
  , div [] [text<|String.fromInt model.count]
  , button [onClick Decrement] [text "-1"]
main = Browser.sandbox
 { init = initialModel
  , view = view
```

String.join ", " < List.sort names Tasks can be chained. Cmds only batched.

JavaScript Interop

port prices : (Float -> msg) -> Sub msg

Pipe Operator

String.join ", " (List.sort names)

Ports, incoming and outgoing values:

port time : Float -> Cmd msg

viewNames1 names =

viewNames2 names =

viewNames3 names =

|> List.sort

|> String.join ", "

names

+1

0