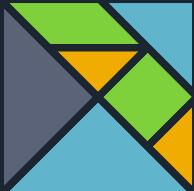


Welcome to Elm!



Please follow these instructions to get set up:

github.com/rtfeldman/elm-0.19-workshop



Introduction

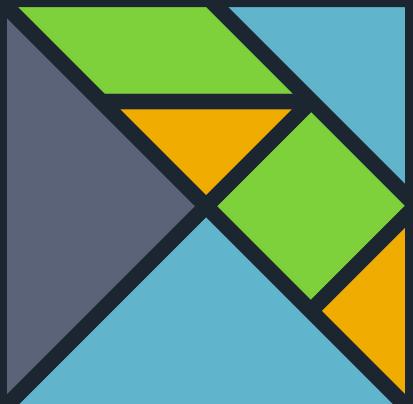
What is Elm?

Who uses Elm?

Costs & Benefits

Workshop Structure

elm



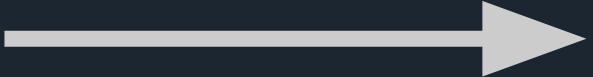
JavaScript



compiles to



BABEL

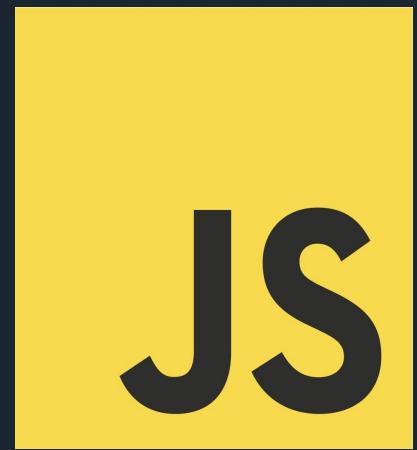


compiles to

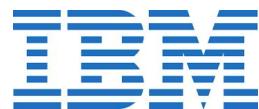
JS



compiles to



Companies of all sizes use Elm



FEATURE
SPACE



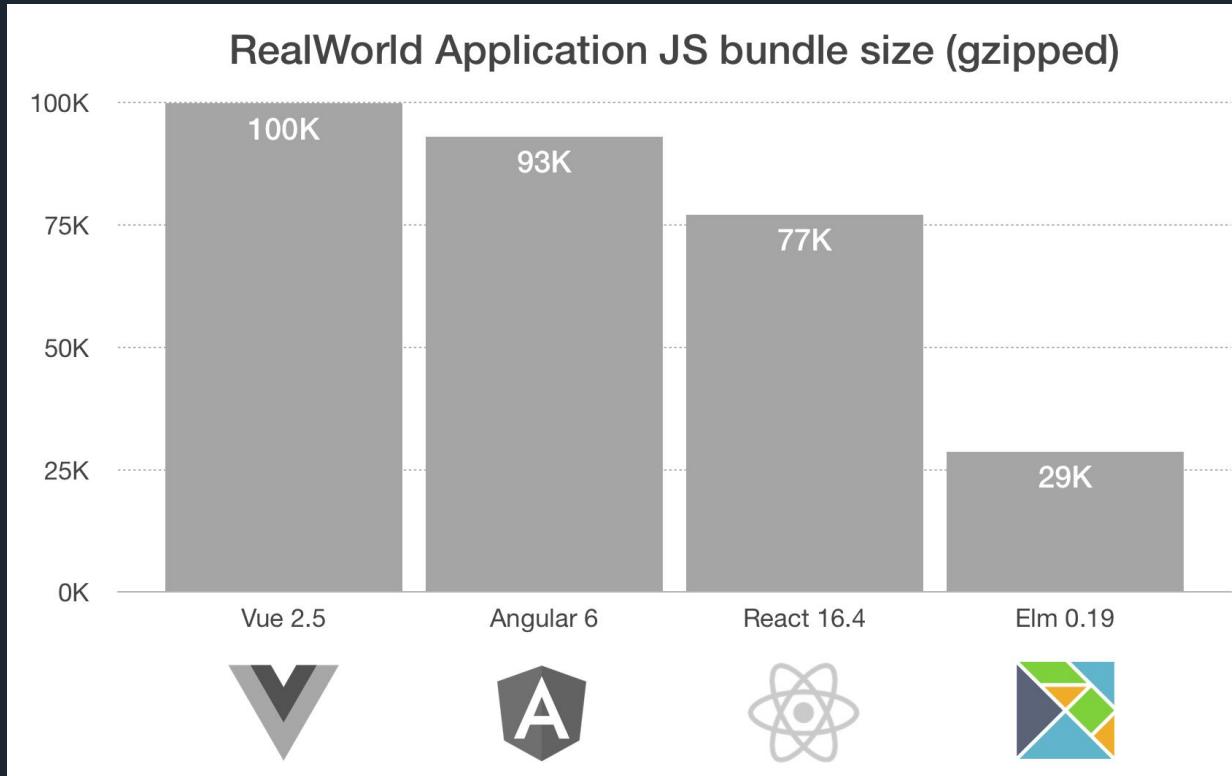
COSTS

1. Learning a new language
2. Smaller ecosystem
3. Fewer Web APIs have first-class support

Why are companies choosing Elm?

1. Measurable technical advantages
2. It makes hiring easier
3. Cohesive, high-quality ecosystem

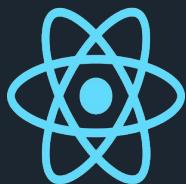
1. Measurable technical advantages



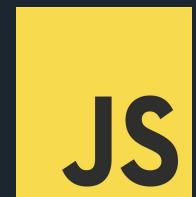
elm-lang.org/blog/small-assets-without-the-headache

1. Measurable technical advantages

React Angular Vue Ember

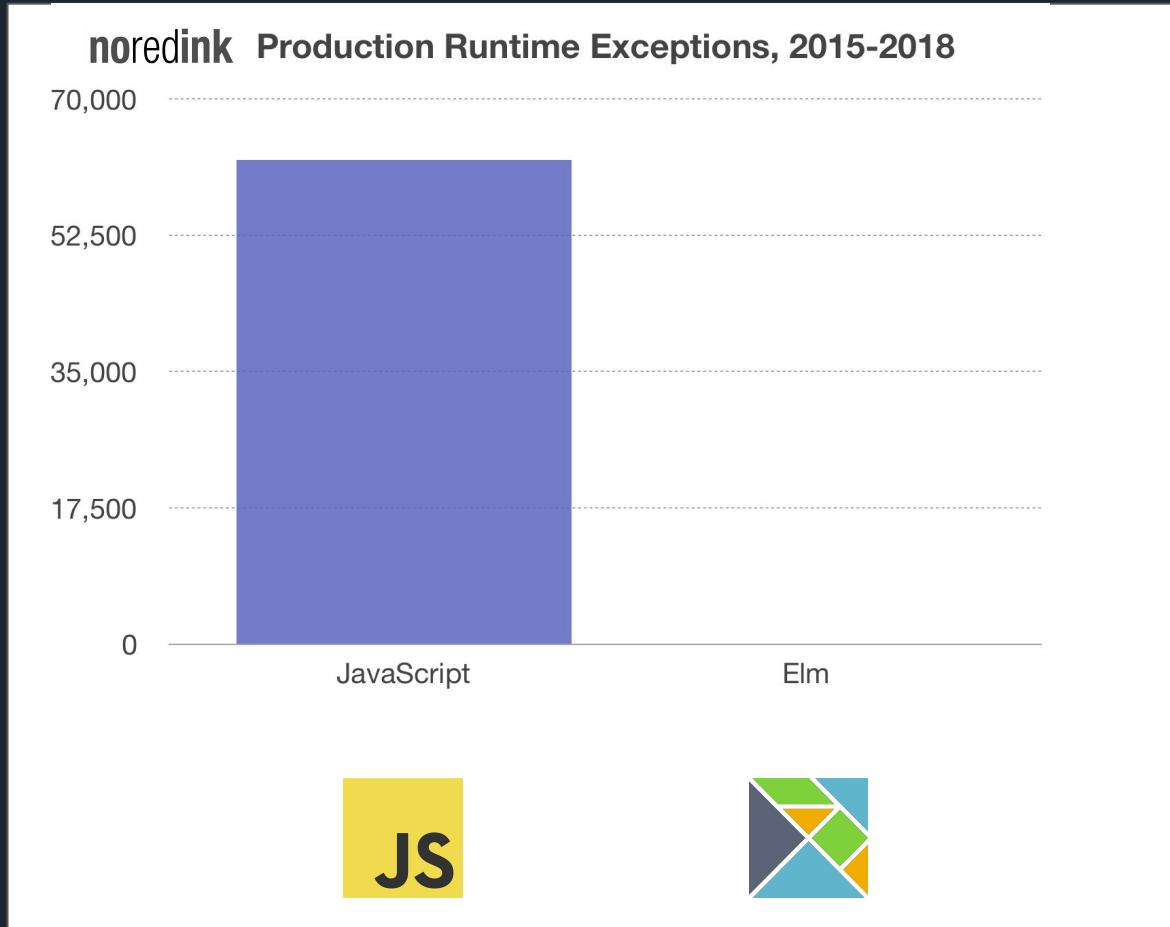


BUNDLE SIZE



Elm Preact Svelte Raw JS

1. Measurable technical advantages



“No runtime exceptions.”

not **zero**,
but negligible

2. It makes hiring easier

noredink **REMOTE POSITIONS**

Elm is **#1 reason** developers apply.

“I have never seen an
inbound pipeline this strong.”

LOCAL POSITIONS



“There is not a large pool of potential candidates for Elm in **Sydney**. However the *quality* of candidates seems to be higher, so we’ve been fortunate enough to fill our current needs.”

ONBOARDING

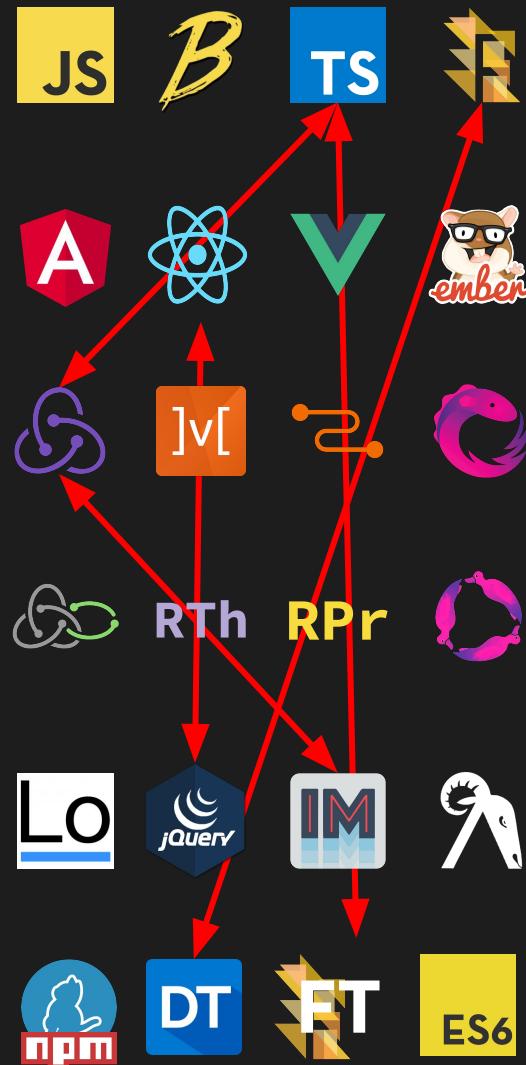
JavaScript bootcamp



no red ink

writing production Elm

FIRST WEEK OF FIRST JOB



Dialect

UI

State

Async

Utilities

Packages



Dialect



view

State

model

Async

update

Utilities

core

Packages

elm install

cohesive ecosystem

fast

reliable

tiny assets

no semicolon debates

Dialect



UI

State

Async

Utilities

Packages

view

model

update

core

elm install

WORKSHOP STRUCTURE

 **RealWorld**
example app

9 lessons
9 exercises

conduit

Home New Post Settings rtfeldman Sign out

conduit

A place to share your knowledge.

Your Feed Global Feed

 testreactthing April 29, 2017 

Wheeeeeee!
This is great stuff
Read more...

 testertestily April 29, 2017 

Here's a BRAND NEW Article!
Everything you ever wanted to know about how to write a fake article.
Read more...

Popular Tags

- dragons
- training
- test
- as
- tags
- coffee
- flowers
- money
- sushi
- cars
- japan
- animation
- cookies
- well
- sugar
- happiness
- baby
- caramel
- clean



1. Rendering a Page

Compiling

Functions

if-expressions

Virtual DOM



```
$ elm make Main.elm --output elm.js
```

Dependencies loaded from local cache.

Dependencies ready!

Success! Compiled 1 module.

index.html

```
<head>
  <link rel="stylesheet" href="main.css">
  <script src="elm.js"></script>
</head>
```

```
<head>
  <link rel="stylesheet" href="main.css">
  <script src="elm.js"></script>
</head>
<body>
  <div id="app"></div>
  <script>
    Elm.Main.init({
      node: document.getElementById("app")
    });
  </script>
</body>
```

INCREMENTAL ADOPTION

functions

User Interface Example

LEAF SEARCH

Your search returned 3 leaves:



Elm leaf



Maple leaf



Oak leaf

Your search returned 2 leaves:



Elm leaf



Maple leaf

Your search returned 1 leaves:



Elm leaf

Your search returned 1 leaves:



Elm leaf

```
console.log(pluralize("leaf", "leaves", 1));
```

leaf

```
let pluralize =
```

```
console.log(pluralize("leaf", "leaves", 3));  
leaves
```

```
let pluralize =  
(singular, plural, quantity) => {  
  if (quantity === 1) {  
    return singular;  
  } else {  
    return plural;  
  }  
};
```

BABEL

```
let pluralize =  
  (singular, plural, quantity) => {  
    if (quantity === 1) {  
      return singular;  
    } else {  
      return plural;  
    }  
  };
```

BABEL

```
var pluralize = this is all the browser ever sees!
function pluralize(
  singular, plural, quantity) {
  if (quantity === 1) {
    return singular;
  } else {
    return singular;
  }
};
```



```
pluralize singular plural quantity =  
    if quantity == 1 then  
        singular  
  
    else  
        plural
```

arguments

pluralize singular plural quantity =

if quantity == 1 then

singular

required

no parentheses

else

plural

```
x = if quantity == 1 then  
    singular  
  
else  
    plural
```

expression

```
x = (quantity === 1 ? singular : plural)
```

JS

```
pluralize singular plural quantity =  
  if quantity == 1 then  
    singular
```

```
else  
  plural
```

```
main = call pluralize passing 3 arguments
```

```
text (pluralize "leaf" "leaves" 1)
```

```
pluralize singular plural quantity =  
  if quantity == 1 then  
    singular
```

```
else  
  plural
```

```
main = call text passing 1 argument
```

```
text (pluralize "leaf" "leaves" 1)
```

```
{- this comment can
   span multiple lines! -}
pluralize singular plural quantity =
  if quantity == 1 then
    singular
  else
    plural

main = -- this is an inline comment
      text (pluralize "leaf" "leaves" 1)
```

js

```
pluralize("leaf", "leaves", 1)      leaf
```

```
let pluralize =  
  (singular, plural, quantity) => {  
    if (quantity === 1) {  
      return singular;  
    } else {  
      return plural;  
    }  
  };
```

js

```
pluralize("leaf", "leaves", 3)      leaves
```

```
let pluralize =  
  (singular, plural, quantity) => {  
    if (quantity === 1) {  
      return singular;  
    } else {  
      return plural;  
    }  
  };
```

js

```
pluralize("leaf", "leaves", 3)      leaves
```

```
let pluralize =  
  (singular, plural, quantity) => {  
    if (quantity === 1) {  
      return singular;  
    } else {  
      return plural;  
    }  
  };
```

js

```
pluralize("leaf", "leaves", 1)
```

Uncaught ReferenceError: singula is not defined

```
let pluralize =  
  (singular, plural, quantity) => {  
    if (quantity === 1) {  
      return singula;  
    } else {  
      return plural;  
    }  
  };
```

```
pluralize("leaf", "leaves", 1)
```

```
let pluralize =  
(singular, plural, quantity) => {  
  if (quantity === 1) {  
    return singula;  
  } else {  
    return plural;  
  }  
};
```

error TS2552:
Cannot find name 'singula'.
Did you mean 'singular'?



```
pluralize "leaf" "leaves" 1
```

```
pluralize singular plural quantity =
```

```
  if quantity == 1 then  
    singular
```

```
  else
```

```
    plural
```



```
pluralize "leaf" "leaves" 1
```

```
pluralize singular plural quantity =
```

```
if quantity == 1 then  
    singula
```

Cannot find variable **singula**.

```
else
```

```
    plural
```

Maybe you want one of the following?

singular

js

```
pluralize("leaf", "leaves", 3)      leaves
```

```
let pluralize =  
  (singular, plural, quantity) => {  
    if (quantity === "1") {  
      return singular;  
    } else {  
      return plural;  
    }  
  };
```

js

```
pluralize("leaf", "leaves", 1) leaves
```

```
let pluralize =  
  (singular, plural, quantity) => {  
    if (quantity === "1") {  
      return singular;  
    } else {  
      return plural;  
    }  
  };
```

```
pluralize("leaf", "leaves", 1) leaves
```

```
let pluralize =  
  (singular, plural, quantity) => {  
    if (quantity === "1") {  
      return singular;  
    } else {  
      return plural;  
    }  
  };
```



```
pluralize "leaf" "leaves" 1
```

type inference!

```
pluralize singular plural quantity =
```

```
if quantity == "1" then  
    singular
```

```
else
```

```
    plural
```

The 3rd argument to function
pluralize is causing a mismatch.

Function **pluralize** is expecting the
3rd argument to be: **String**

But it is: **number**

What we'll be building

conduit

Home New Post Settings rtfeldman Sign out

conduit

A place to share your knowledge.

Your Feed Global Feed

 testreactthing April 29, 2017

Wheeeee!
This is great stuff

Read more...

 testertestily April 29, 2017

Here's a BRAND NEW Article!
Everything you ever wanted to know about how to write a fake article.

Read more...

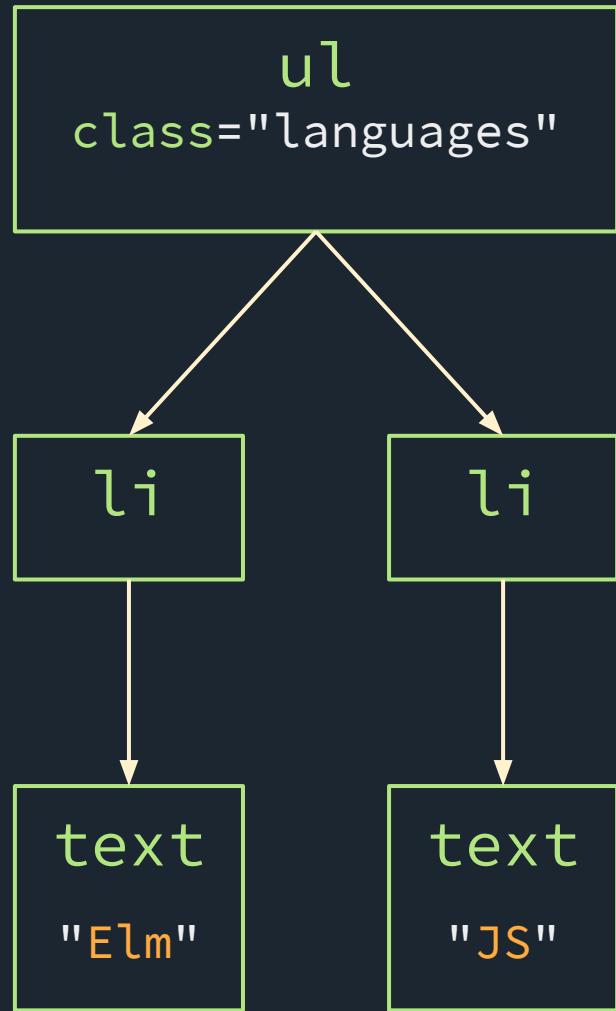
Popular Tags

- dragons
- training
- test
- as
- tags
- coffee
- flowers
- money
- sushi
- cars
- japan
- animation
- cookies
- well
- sugar
- happiness
- baby
- caramel
- clean

Virtual DOM

```
<ul class="languages">  
  <li>Elm</li>  
  <li>JS</li>  
</ul>
```

```
ul [ class "languages" ] [  
  [ li [] [ text "Elm" ],  
  , li [] [ text "JS" ]  
]
```



```
li [] [ text "Elm" ] li [] [ text "JS" ]
```

error: passing 5 arguments to `li`!

```
ul [ class "languages" ] [  
  li [] [ text "Elm" ] ,  
  li [] [ text "JS" ]  
]
```



```
ul [ class "languages" ]
[ li [] [ text "Elm" ]
, li [] [ text "JS" ]
]
```

```
ul [ class "languages" ]
[ li [] [ text "Elm" ]
, li [] [ text "JS" ]
, li [] [ text "Bash" ]
]
```



Review of Part 1

Compiling

```
elm make Main.elm --output elm.js
```

Functions

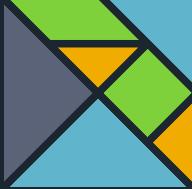
```
pluralize "leaf" "leaves" 1
```

if-expressions

```
if quantity == 1 then
```

Virtual DOM

```
li [] [ text "Elm" ]
```



Exercises for Part 1

conduit

Home Sign in Sign up

TODO: Replace this <p> with the banner
(In the future we'll display a feed of articles here!)

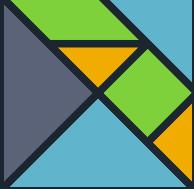


conduit

Home Sign in Sign up

conduit
A place to share your knowledge.
(In the future we'll display a feed of articles here!)

Follow the instructions in **part1/README.md**



2. Manipulating Values

Strings

let-expressions

Lists

Partial application

strings

"foo"  "bar" → "foobar"



"foo"  "bar" → "foobar"

```
pluralize singular plural quantity =  
if quantity == 1 then  
    quantity ++ " " ++ singular  
else  
    quantity ++ " " ++ plural
```

```
pluralize singular plural quantity =  
if quantity == 1 then  
    String.fromInt quantity ++ " " ++ singular  
else  
    String.fromInt quantity ++ " " ++ plural
```

code duplication

let-expressions

pluralize singular plural quantity =

```
let
    quantityStr =
        String.fromInt quantity

    prefix =
        quantityStr ++ " "
in
if quantity == 1 then
    prefix ++ singular
else
    prefix ++ plural
```

let-expression

```
pluralize singular plural quantity =
```

```
let
```

```
    quantityStr =
```

```
        String.fromInt quantity
```

accessible inside *let*

```
    prefix =
```

```
        quantityStr ++ " "
```

```
in
```

```
if quantity == 1 then
```

```
    prefix ++ singular
```

```
else
```

```
    prefix ++ plural
```

```
pluralize singular plural quantity =  
let  
  quantityStr =  
inaccessible in  String.fromInt quantity  
outside scope  
  prefix =  
    quantityStr ++ " "  
in  
if quantity == 1 then  
  prefix ++ singular  
  
else  
  prefix ++ plural
```

```
pluralize singular plural quantity =  
let  
    quantityStr =  
        String.fromInt quantity  
  
    prefix =  
        quantityStr ++ " "  
in  
    if quantity == 1 then  
        prefix ++ singular  
  
    else  
        prefix ++ plural
```

entire expression
evaluates to this

```
pluralize singular plural quantity =
```

```
let
```

```
    quantityStr =
```

```
must have same String.fromInt quantity  
indentation level
```

```
    prefix =
```

```
        quantityStr ++ " "
```

```
in
```

```
if quantity == 1 then
```

```
    prefix ++ singular
```

```
else
```

```
    prefix ++ plural
```

lists

```
[ 1, 2, 3 ]
```

```
[ [ "foo", "bar" ], [ "baz" ] ]
```

```
[ "foo", 65 ]
```

```
["pow", "zap", "blam"].map(  
  (str) => { return str.toUpperCase() + "!"; }  
)
```

```
[ "POW!", "ZAP!", "BLAM!" ]
```

```
["pow", "zap", "blam", 500].map(  
  (str) => { return str.toUpperCase() + "!"; }  
)
```

Uncaught TypeError: str.toUpperCase is not a function

solution:

require **consistent** entries

```
["pow", "zap", "blam", 500].map(  
  (str) => { return str.toUpperCase() + "!"; }  
)
```

anonymous function

```
List.map (\str -> String.toUpperCase str ++ "!")
[ "pow", "zap", "blam" ]
```

λ

uppercase the string

```
List.map (\str -> String.toUpperCase str ++ "!")  
[ "pow", "zap", "blam" ]  
[ "POW!", "ZAP!", "BLAM!" ]
```

```
List.map (\str -> String.toUpperCase str ++ "!")  
[ "pow", "zap", "blam", 500 ]
```

The 3rd and 4th entries in this list are different types of values.

Hint: Every entry in a list needs to be the same type of value.

partial application

```
List.map (\num -> pluralize "leaf" "leaves" num)  
[ 0, 1, 2 ]
```

```
[ "0 leaves", "1 leaf", "2 leaves" ]
```

```
List.map (pluralize "leaf" "leaves") [ 0, 1, 2 ]  
(\num -> pluralize "leaf" "leaves" num)
```

List.map for rendering

```
names = [ "Erica", , "Shuri", , "Lee" ]
```

ul []

→ [li [] [text "Erica"]
→ , li [] [text "Shuri"]
→ , li [] [text "Lee"]
]

```
viewName name =  
    li [] [ text name ]
```

```
ul [] (List.map viewName names)
```



Review of Part 2

Strings

```
"d" ++ String.fromInt 6 → "d6"
```

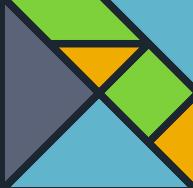
let-expressions

```
let      x = 5      in      x * 2
```

Lists

```
List.map (\x -> x + 1) nums
```

Partial application pluralize "leaf" "leaves"



Exercises for Part 2

ledge.

Popular Tags

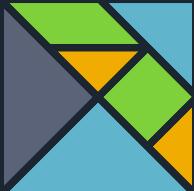


ledge.

Popular Tags

elm fun programming
compilers

Follow the instructions in [part2/README.md](#)



3. Introducing Interaction

Records

Record Updates

Booleans

The Elm Architecture

records

record = plain, immutable data
{ name = "foo", x = 1, y = 3 }

no prototypes

no **this**

no mutating

record.name → "foo"

record.x → 1

record.y → 3

```
record =  
  { name = "foo", x = 1, y = 3 }
```

```
newRecord =  
  { record | name = "bar" }
```

```
newRecord =  
  { name = "bar"  
  , x = 1  
  , y = 3  
  }
```

iteration

[1, 2, 3]

map works fine

[1, 2, "foo"]

map would break

{ x = 1, y = "foo" }

iteration

mixed entries

lists

supported

unsupported

records

unsupported

supported

booleans

True False

x == y

not (x == y)

x /= y

x || y

x && y

booleans for list membership

List.member 1 [1, 2, 3]

True

List.member 9 [1, 2, 3]

False

booleans for filtering lists

```
[ 1, 2, 3 ]
```

```
isKeepable num =  
    num > 1
```

```
List.filter isKeepable [ 1, 2, 3 ]
```

```
[ 2, 3 ]
```

```
\num -> num > 1
```

```
isKeepable num =  
    num > 1
```

```
List.filter (\num -> num > 1) [ 1, 2, 3 ]  
[ 2, 3 ]
```

conduit

A place to share your knowledge.

Elm is fun!

Elm

[Read more...](#)

Popular Tags

elm fun programming
dragons

Who says undefined isn't a function anyway?

Functions

[Read more...](#)

This compiler is pretty neat

Elm

[Read more...](#)

conduit

A place to share your knowledge.

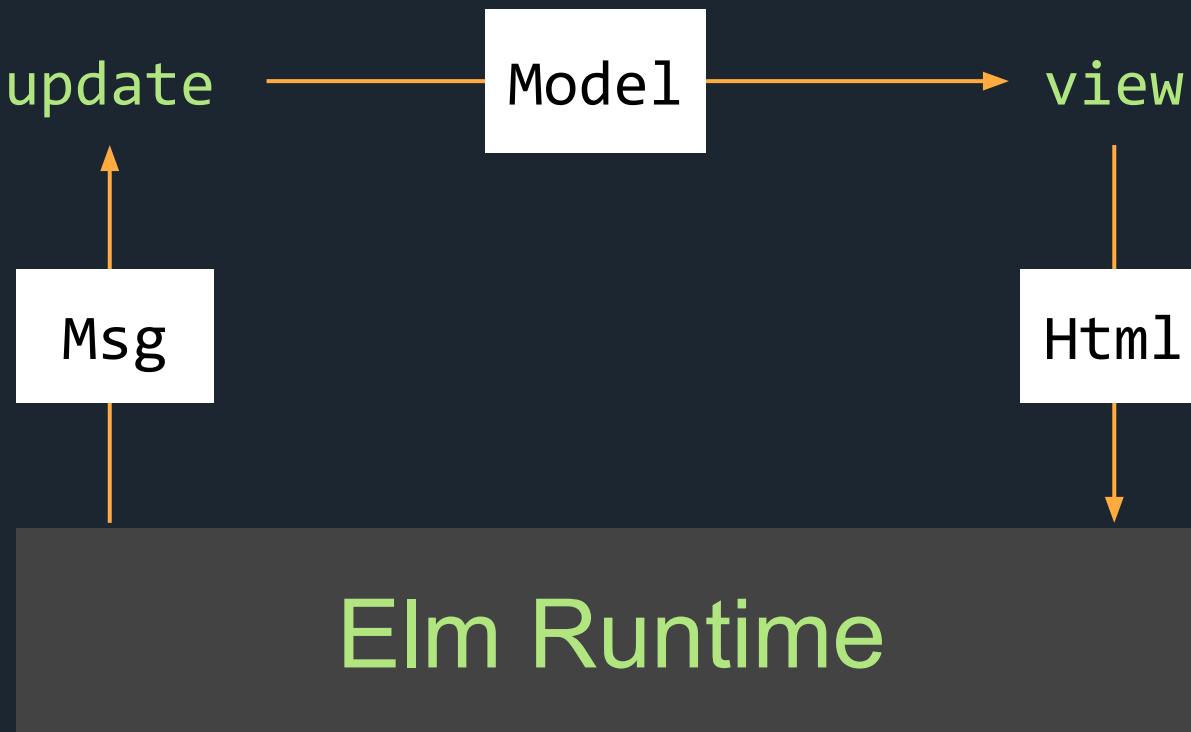
Who says undefined isn't a function anyway?

Functions

Read more...

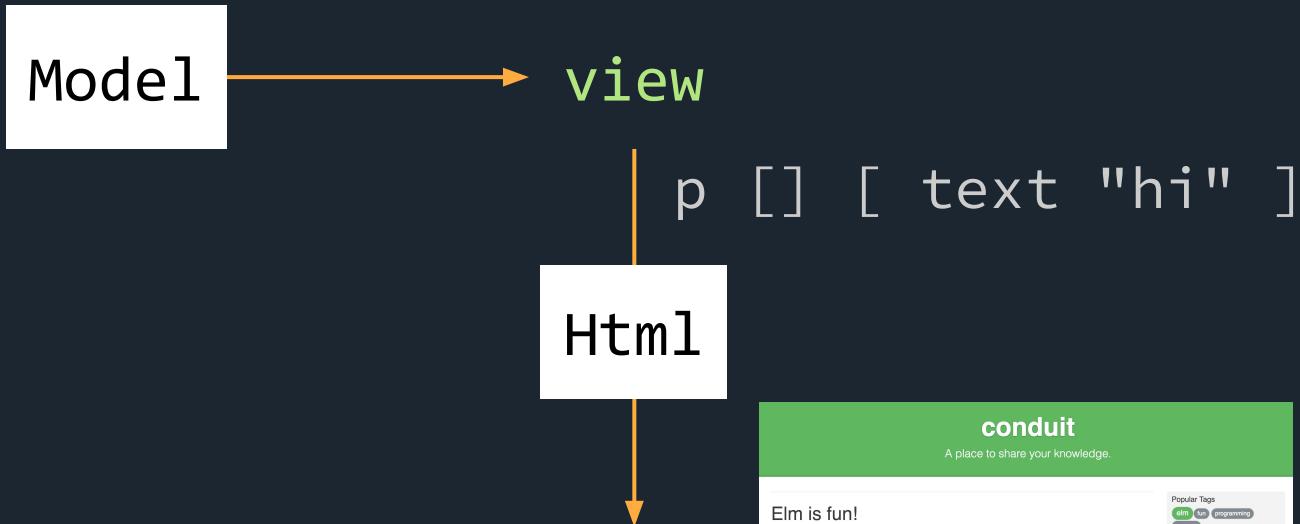


The Elm Architecture



entire application state

view **model** =
p [] [text "hi"]



Elm Runtime

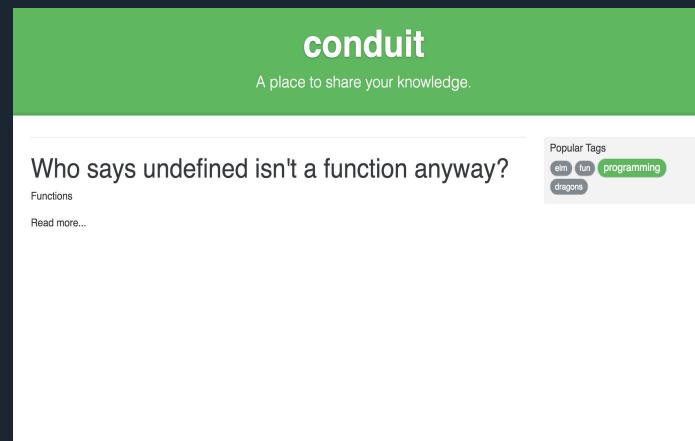
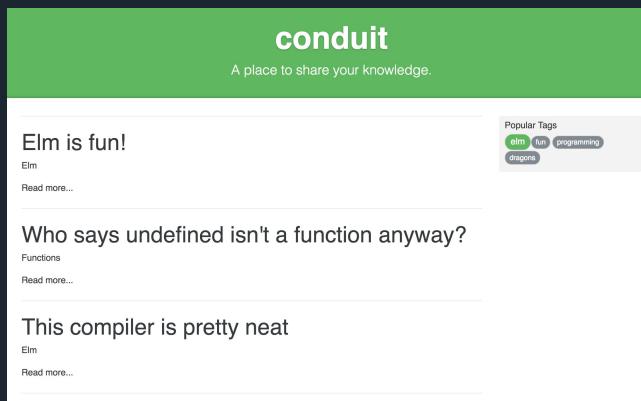
A screenshot of the [conduit](#) website, which is described as "A place to share your knowledge". The page features a green header bar with the word "conduit" and a sub-header "A place to share your knowledge.". Below the header, there are three blog post cards:

- Elm is fun!**
Elm
Read more...
- Who says undefined isn't a function anyway?**
Functions
Read more...
- This compiler is pretty neat**
Elm
Read more...

At the bottom right of the screenshot, there is a "Popular Tags" section with several colored tags: elm, fun, programming, and dragons.

view model = ...

initial Model → Model with different tag selected



```
update msg model =  
{ model | selectedTag = "elm" }
```

returns a new model

```
update msg model =
```

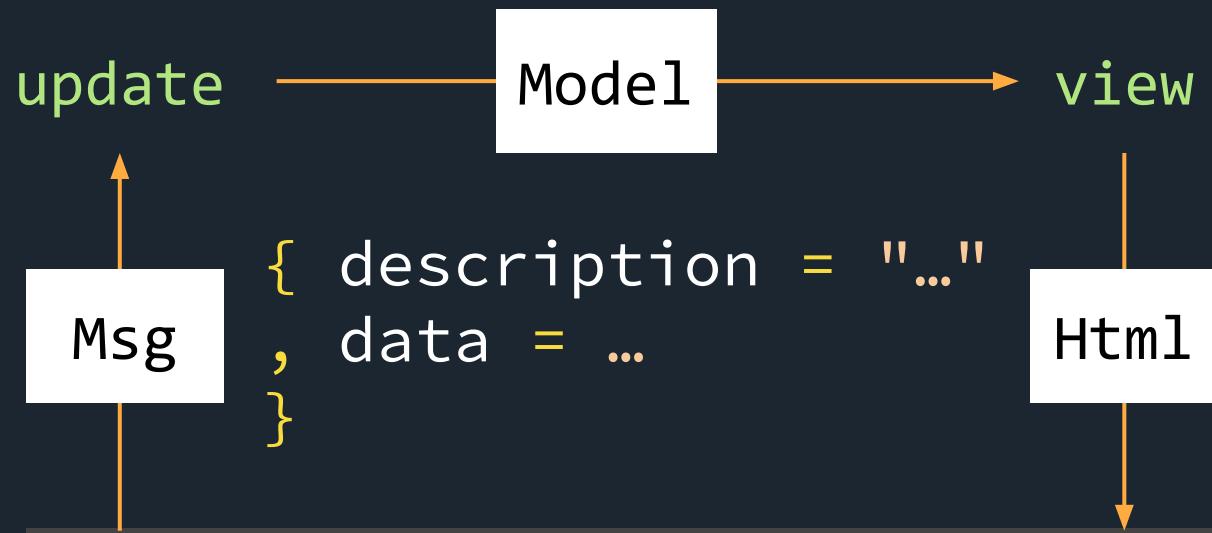
```
button
```

```
[ onClick
```

```
{ description = "ClickedTag"  
, data = "elm"  
}
```

```
]
```

```
[ text "elm" ]
```



Elm Runtime

The screenshot shows a blog post on the **conduit** website:

- conduit** (header)
- A place to share your knowledge.
- Who says undefined isn't a function anyway?** (post title)
- Functions (category)
- Read more... (button)
- Popular Tags: elm, fun, programming, dragons (tags)



Review of Part 3

Records

```
foo = { bar = "baz", x = 5 }
```

Record Updates

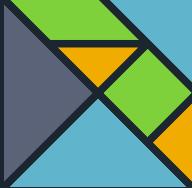
```
{ foo | x = 42 }
```

Booleans

```
not (foo || bar) && True
```

The Elm Architecture

```
model, view, update
```



Exercises for Part 3

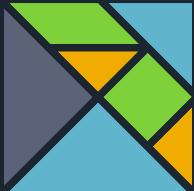
The diagram illustrates the progression of a user's work on the Conduit platform. It consists of two side-by-side screenshots of a web application interface.

Left Screenshot: This shows a list of posts on the Conduit homepage. The first post is titled "Elm is fun!" and includes a snippet of Elm code. The second post is titled "Who says undefined isn't a function anyway?" and includes a snippet of JavaScript code. The third post is titled "This compiler is pretty neat" and includes a snippet of Elm code. Each post has a "Read more..." link below it. A "Popular Tags" section at the bottom right lists "elm", "fun", "programming", and "dragons".

Right Screenshot: This shows the same list of posts, but the second post has been updated. Its title is now "Who says undefined isn't a function anyway?" and its snippet of code has been modified to use the "Function" keyword instead of "function". The other posts and the "Popular Tags" section remain the same.

An orange arrow points from the left screenshot to the right one, indicating the flow of the exercise or the progression of the user's work.

Follow the instructions in [part3/README.md](#)



4. Type Annotations

Primitives

Parameters

Aliases

Functions

```
-- username is a string
```

```
username = "rtfeldman"
```

```
username : String
```

```
username = "rtfeldman"
```

```
totalPages : Int
```

```
totalPages = 5
```

```
isActive : Bool
```

```
isActive = True
```

round : Float -> Int

not : Bool -> Bool

String.length : String -> Int

`:` is for **types**

```
searchResult : { name : String, stars : Int }  
searchResult = { name = "blah", stars = 215 }
```

`=` is for **values**

Parameterized Type

Type Parameter

```
names : List String
```

```
names = [ "Sam", "Casey", "Pat" ]
```

“List of Strings”

names : List String

names = ["Sam", "Casey", "Pat"]

```
names : List Bool  
names = [ True, True, False ]
```

```
names : List Float  
names = [ 1.1, 2.2, 3.3 ]
```

```
names : List Int    ERROR!  
names = [ 1.1, 2.2, 3.3 ]
```

```
model :  
  { selectedTag : String  
  , articles :  
    List  
      { title : String  
      , tags : List String  
      , body : String  
      }  
  }
```

```
type alias Article =  
    { title : String  
    , tags : List String  
    , body : String  
    }
```

```
type alias Model =  
    { selectedTag : String  
    , articles : List Article  
    }
```

```
type alias Msg =  
    { description : String  
    , data : String  
    }
```

```
view model =  
    button  
        [ onClick  
            { description = "ClickedClear"  
            , data = "ALL"  
            }  
        ]  
        [ text "X" ]
```

```
view : Model -> Html String
view model =
    button [ onClick "ClickedClearAll" ] [ text "X" ]
```

```
view : Model -> Html Msg
view model =
    button
        [ onClick
            { description = "ClickedClear"
            , data = "ALL"
            }
        ]
        [ text "X" ]
```

```
view : Model -> Html String  
view model =  
    button [ onClick "..." ] [ text "X" ]
```

```
view : Model -> Html Msg  
view model =  
    button [ onClick { ... } ] [ text "X" ]
```

```
view : Model -> Html Float  
view model =  
    button [ onClick 2.6 ] [ text "X" ]
```

```
type alias Msg =  
    { description : String  
    , data : String  
    }
```

```
view : Model -> Html Msg  
view model =
```

```
update : Msg -> Model -> Model  
update msg model =
```

elm repl

```
> pluralize "leaf" "leaves" 3  
"3 leaves" : String  
  
> pluralize  
String -> String -> Int -> String  
  
> pluralize "leaf" "leaves"  
Int -> String
```

(pluralize "leaf" "leaves") 3 String

(pluralize "leaf" "leaves") Int -> String

(pluralize "leaf" "leaves") 3 String

```
pluralize : String -> String -> Int -> String
```

```
(pluralize)      String -> (String -> (Int -> String))
```

```
(pluralize "leaf")      String -> (Int -> String)
```

```
((pluralize "leaf") "leaves")      Int -> String
```

```
((pluralize "leaf") "leaves") 3      String
```

update : Msg -> Model -> Model



Review of Part 4

Annotations

```
point : { x : Int, foo : Bool }
```

Parameters

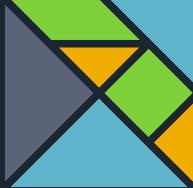
```
List String           Html Msg
```

Aliases

```
type alias Model = { ... }
```

Functions

```
update : Msg -> Model -> Model
```



Exercises for Part 4

```
{-| ↗ TODO: Replace this comment with a type annotation for `update`  
-}  
update msg model =  
  if msg.description == "ClickedTag" then  
    { model | selectedTag = msg.data }  
  
  else  
    model
```

Follow the instructions in [part4/README.md](#)



5. Custom Types

case-expressions

Enumerations

Containers

Variant Functions

ADDING FEATURES TO THE FEED



kaka111

August 10, 2018

0

a

a

Read more...

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	
42	43	44	45	46	47	48	49	50												

PAGINATION

TABS

Your Feed

Global Feed

#happiness



title

nothing

[Read more...](#)



Popular Tags

dragons training test tags as
coffee flowers money
happiness sushi caramel sugar
japan cars cookies well
animation clean baby

Your Feed

Global Feed

#happiness

```
type alias Model =  
    { tab : String  
    , ...  
    }
```

```
yourFeed      = "YourFeed"  
globalFeed   = "GlobalFeed"  
tagFeed       = "TagFeed"
```

```
if model.tab == "YourFeed" then
    -- show Your Feed

else if model.tab == "GlobalFeed" then
    -- show Global Feed

else
    -- show Tag Feed
```

```
case model.tab of
    "YourFeed" ->
        -- show Your Feed
    "GlobalFeed" ->
        -- show Global Feed
    _ ->
        -- show Tag Feed
```

must have same
indentation level

also known in other languages as...

“sum types” “union types” “ADTs”

custom type

type Tab = variants

YourFeed | GlobalFeed | TagFeed

yours : Tab

yours =

YourFeed

global : Tab

global =

GlobalFeed

tag : Tab

tag =

TagFeed

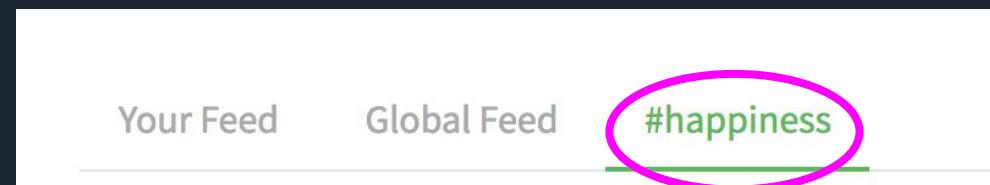
```
type Tab  
  = YourFeed  
  | GlobalFeed  
  | TagFeed
```

```
type Bool custom type  
  = True variants  
  | False
```

CUSTOM TYPE

```
type Tab  
= YourFeed  
| GlobalFeed  
| TagFeed
```

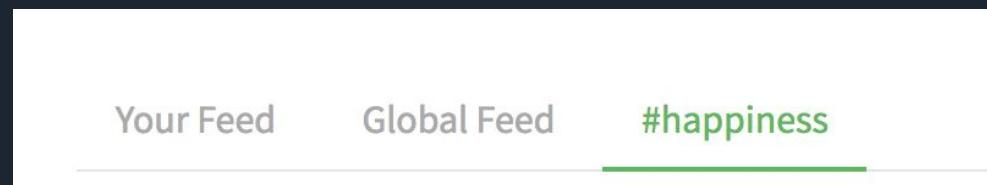
```
type alias Model =  
{ tab : Tab  
, selectedTag : String  
, ...  
}
```



```
case model.tab of  
YourFeed ->  
-- show your feed  
  
GlobalFeed ->  
-- show global feed  
  
TagFeed ->  
-- show tag feed  
  
no default case!
```

```
type Tab  
= YourFeed  
| GlobalFeed  
| TagFeed String
```

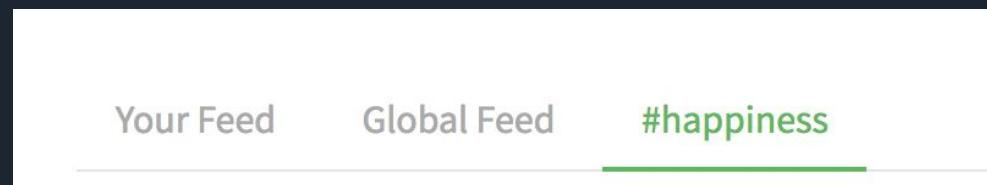
```
> TagFeed "happiness"  
TagFeed "happiness" : Tab  
  
> TagFeed  
<function> : String -> Tab
```



```
yours : Tab  
yours =  
  YourFeed  
  
happiness : Tab  
happiness =  
  TagFeed "happiness"
```

```
type Tab  
  = YourFeed  
  | GlobalFeed  
  | TagFeed String
```

~~-enumeration~~
container



```
case model.tab of  
  YourFeed ->  
    -- show your feed  
  
  GlobalFeed ->  
    -- show global feed  
  
  TagFeed selectedTag ->  
    -- show tag feed
```



kaka111

August 10, 2018



a

a

Read more...

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	
42	43	44	45	46	47	48	49	50												

PAGINATION

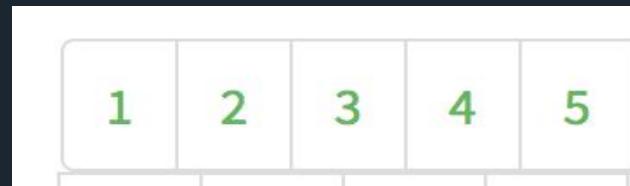
```
type alias Msg =  
    { description : String  
    , data : String  
    }
```

```
{ description = "ClickedTag"  
, data = "cars"
```

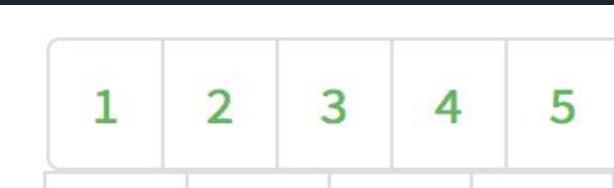
```
{ description = "ClickedPage"  
, data = 2
```

Popular Tags

dragons training test tags as
coffee flowers sugar money
happiness sushi caramel
cookies cars japan well
animation clean baby



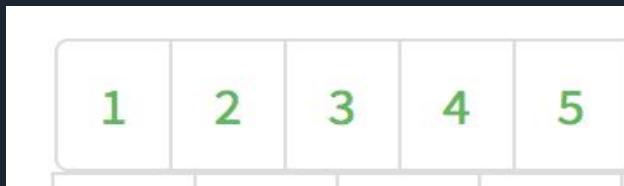
```
type alias Msg =  
    { description : String  
    , stringData : String  
    , intData : Int  
    }  
  
{ description = "ClickedTag"  
, data = "cars"  
}  
  
{ description = "ClickedPage"  
, data = 2  
}
```



```
type alias Msg =  
    { description : String  
    , stringData : String  
    , intData : Int  
    }  
  
{ description = "ClickedTag"  
, stringData = "cars"  
, intData = -1  
}  
  
{ description = "ClickedPage"  
, stringData = ""  
, intData = 2  
}
```

Popular Tags

dragons training test tags as
coffee flowers sugar money
happiness sushi caramel
cookies cars japan well
animation clean baby



```
type alias Msg = { description : String, stringData : String, intData : Int }  
  
{ description = "ClickedTag"    clickedTag "cars"  
, stringData = "cars"  
, intData = -1  
}  
  
{ description = "ClickedPage"   clickedPage 2  
, stringData = ""  
, intData = 2  
}
```

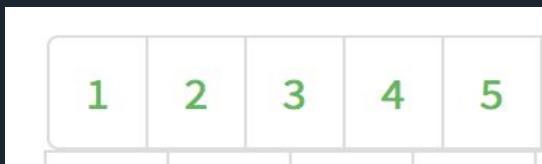
```
type Msg  
= ClickedTag String  
| ClickedPage Int
```

```
type Msg  
  = ClickedTag String  
  | ClickedPage Int
```

```
update msg model =  
  case msg of  
    ClickedTag selectedTag ->  
      -- use tag here  
  
    ClickedPage page ->  
      -- use page here
```

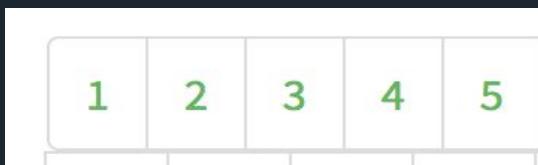
```
type Msg  
  = ClickedTag String  
  | ClickedPage Int
```

```
pageButton : Int -> Html Msg  
pageButton pageNumber =  
  button [ onClick { description = ... } ]
```



```
type Msg  
    = ClickedTag String  
    | ClickedPage Int
```

```
pageButton : Int -> Html Msg  
pageButton pageNumber =  
    button [ onClick (clickedPage pageNumber) ]  
        [ text (String.fromInt pageNumber) ]
```





Review of Part 5

case-expressions

```
case msg of
```

Enumerations

```
type Bool = True | False
```

Containers

```
type Msg = ClickedPage Int | ...
```

Variant Functions

```
onClick (ClickedPage pageNumber)
```



Exercises for Part 5

Sign up

[Have an account?](#)

[Sign up](#)

Follow the instructions in [part5/README.md](#)



6. Maybe

List.head

Maybe

Type Variables

Pipelines

```
first = (users) => {  
    return users[0];  
}  
  
first(["Sam", "Jess"])
```

"Sam"

```
first([])
```

undefined

```
first users =  
List.head users
```

```
first ["Sam", "Jess"]
```

Just "Sam"

```
first []
```

Nothing

```
first = (users) => {
  return users[0];
}
```

```
first users =
List.head users
```

"Sam"	Just "Sam"
undefined	Nothing

```
first(newUsers).length
```

3

TypeError: Cannot read
property 'length' of
undefined

```
case first newUsers of
  Just user ->
    String.length user
  Nothing ->
    0
```

List.head ["Sam", "Jess"]	→	Just "Sam"
List String		Maybe String
List.head [3.14, 7.77]	→	Just 3.14
List Float		Maybe Float
List.head [True, False]	→	Just True
List Bool		Maybe Bool
List.head : ???		

Type Variables

List.head : List elem -> Maybe elem

List.head : List val -> Maybe val

List.head : List v -> Maybe v

Type Variables

List.head : List elem -> Maybe elem

List.reverse : List item -> List item

List.reverse [1, 2, 3] → [3, 2, 1]

List.singleton : val -> List val

List.singleton "foo" → ["foo"]

Type Variables

List.head : List elem -> Maybe elem

```
type Maybe val
= Just val
| Nothing
```

PIPELINES

```
List.reverse (List.filter (\x -> x < 5) [ 2, 4, 6 ])
```

```
[ 2, 4, 6 ]
```

```
|> List.filter (\x -> x < 5)  
|> List.reverse
```

```
List.reverse (List.filter (\x -> x < 5) [ 2, 4, 6 ])
```

```
[ 2, 4, 6 ]
```

```
|> List.filter (\x -> x < 5)  
|> List.reverse  
|> List.map negate
```

```
List.reverse (List.filter (\x -> x < 5) [ 2, 4, 6 ])
```

```
[ 2, 4, 6 ]
```

```
| > List.filter (\x -> x < 5)  
| > List.reverse  
| > List.map negate  
| > List.head
```

Pipelines



Review of Part 6

List.head

```
List.head [ 1, 2, 3 ] → Just 1
```

Maybe

```
type Maybe v = Just v | Nothing
```

Type Variables

```
head : List elem -> Maybe elem
```

Pipelines

```
|> List.head
```



Exercises for Part 6

conduit

Home New Post Settings  adsfadf Sign out

Article Title

What's this article about?

Write your article (in markdown)

Enter tags

Publish Article

A screenshot of the Conduit web application interface. At the top, there is a navigation bar with links for Home, New Post, Settings, a user profile icon, and Sign out. Below the navigation bar are four input fields: 'Article Title', 'What's this article about?', 'Write your article (in markdown)', and 'Enter tags'. A large green button labeled 'Publish Article' is positioned at the bottom right of the form area.

Follow the instructions in **part6/README.md**



7. Decoding JSON

Decoding

Result

Pipeline Decoding

Optional & Nullable

`parseInt "42"` → 42

`parseInt "hi"` → NaN



```
case String.toInt str of
  Just num ->
    num * 10

  Nothing ->
    0
```

`String.toInt "42" → Just 42`

`String.toInt "hi" → Nothing`

```
case String.toInt str of      type Maybe val
  Just num ->                  = Just val
    num * 10                   | Nothing
```

`Nothing ->`
`0`

```
case decodeString Json.Decode.int "42" of
```

```
case decodeString int "42" of
  Ok num ->
    -- Do something with num : Int

  Err error ->
    -- Do something with the error
```

<pre>type Maybe val = Just val Nothing</pre>	<pre>type Result okVal errVal = Ok okVal Err errVal</pre>
--	---

```
{  
  "user_id": 27,  
  "first_name": "Al",  
  "last_name": "Kai"  
}
```

JS

`id = json.id`

`undefined`

TS

`id = json.id`

`undefined`

```
{  
  "user_id": 27,  
  "first_name": "Al",  
  "last_name": "Kai"  
}
```

JS

```
id = json.id
```

undefined

TS

```
id : int = json.id
```

undefined

```
{  
  "user_id": 27,  
  "first_name": "Al",  
  "last_name": "Kai"  
}
```

```
type alias User =  
  { id : Int  
  , firstName : String  
  , lastName : String  
 }
```

```
user : Decoder User  
user =  
  Json.Decode.succeed User  
  |> required "user_id" int  
  |> required "first_name" string  
  |> required "last_name" string
```

```
type alias User =  
    { id : Int  
    , firstName : String  
    , lastName : String  
    }
```

```
case decodeString user json of  
    Ok user ->  
        ...  
    Err error ->  
        ...
```

```
string : Decoder String
```

```
int : Decoder Int
```

```
user : Decoder User
```

```
users : Decoder (List User)
```

```
users =
```

```
list user
```

```
{                                     type alias User =  
  "user_id": 27,                      { id : Int  
  "name": null                         , name : Maybe String  
}                                         , email : String  
                                         }  
  
user : Decoder User  
user =  
  Json.Decode.succeed User  
  |> required "user_id" int  
  |> required "name" (nullable string)  
  |> optional "email" string "me@foo.com"
```



Review of Part 7

Decoding

```
decodeString int "5"
```

Result

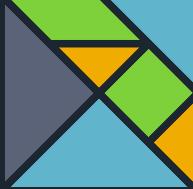
```
type Result x a = Ok a | Err x
```

Pipeline Decoding

```
| > required "id" int
```

Optional & Nullable

```
required "id" (nullable int)
```



Exercises for Part 7

Global Feed

 SamSample
May 5, 2018

(needs decoding!)
(needs decoding!)

[Read more...](#)

 SamSample
May 5, 2018

(needs decoding!)
(needs decoding!)

[Read more...](#)



Global Feed

 SamSample
May 5, 2018

Hello World
stuff

[Read more...](#)

 SamSample
May 5, 2018

this is my title
this is just a test

[Read more...](#)

Follow the instructions in **part7/README.md**



8. Talking to Servers

Tuples

Randomness

Commands

HTTP

TUPLES

serve the same purpose as records

except with **field positions** instead of **field names**

```
x = Tuple.first
```



```
( 5, 7 )
```

```
( name, x, y ) =
```

```
( "foo", 5, 7 )
```

3 elements max!

```
x = { x = 5, y = 7 }.x
```

```
{ name, x, y } = { name = "foo", x = 5, y = 7 }
```

records and tuples are nothing more than
groups of values that travel together

```
animate { name = "foo", x = 1, y = 3 }
```

```
animate ( "foo", 1, 3 )
```

```
animate "foo" 1 3
```

...but **custom types** can be so much more!

function guarantees

same arguments?

same return value

Math.random()

~~Math.random()~~

Random.generate

Random.generate

pickGreeting : List String -> String

Random.generate

~~pickGreeting : List String → String~~

Random.generate

~~pickGreeting : List String → String~~

pickGreeting : List String -> Cmd Msg

```
{ tags = ...  
, selectedTag = ...  
}
```

update

Model

view

Msg

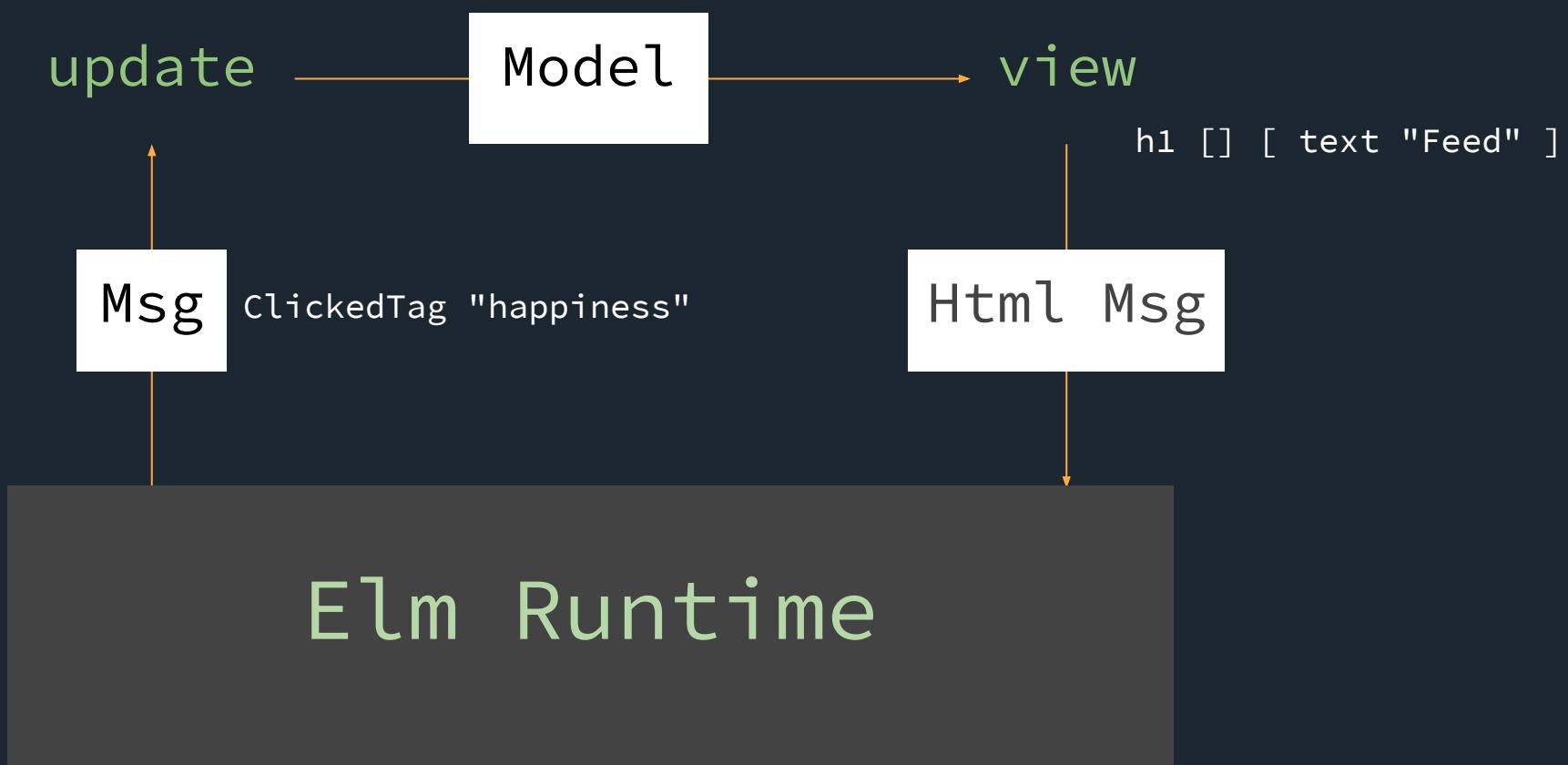
```
{ description = "ClickedTag"  
, data = "happiness"  
}
```

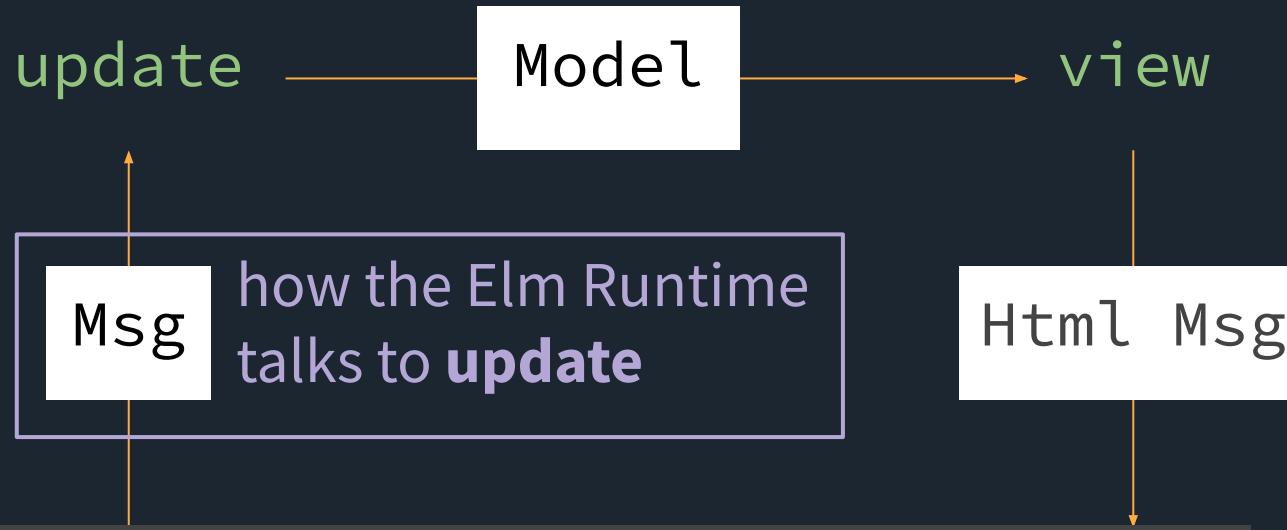
Html

```
h1 [] [ text "Feed" ]
```

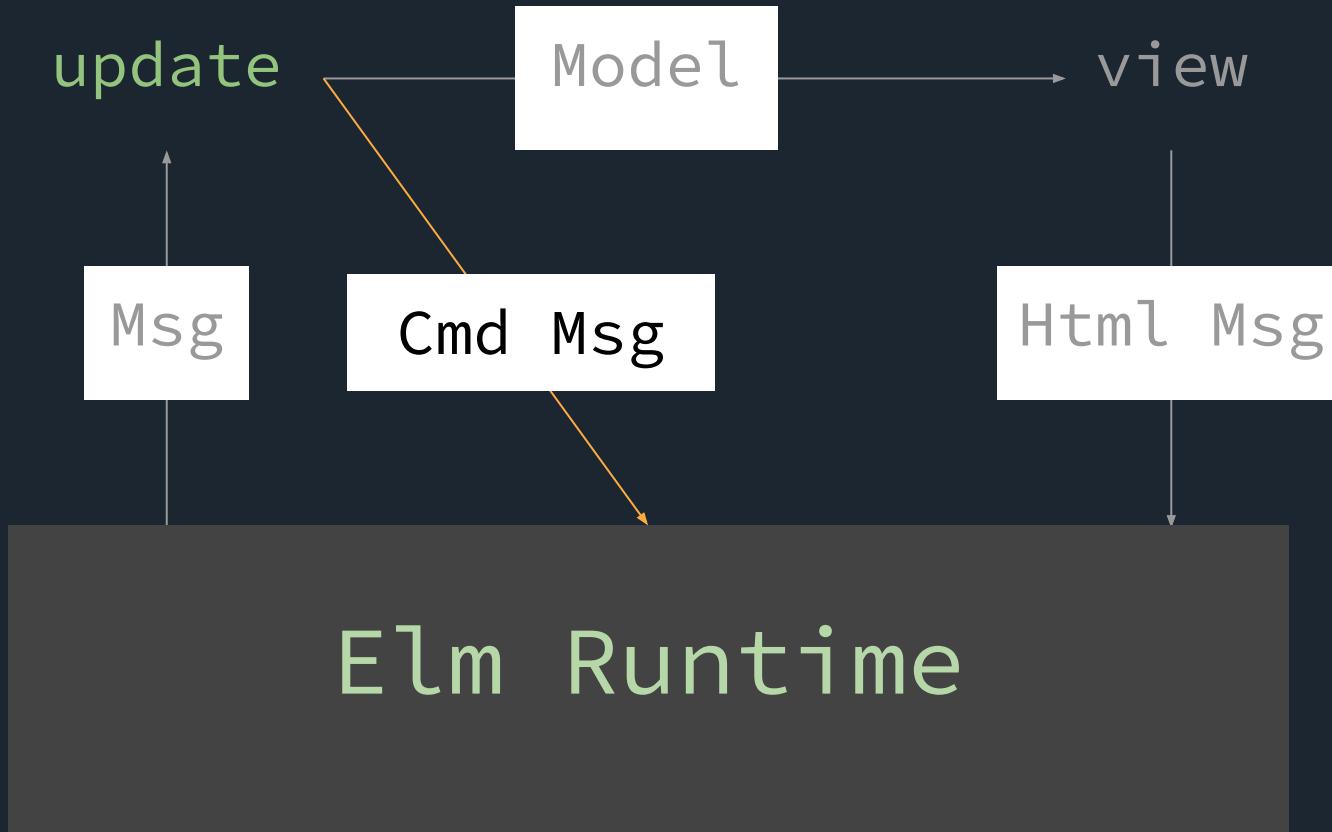
Elm Runtime

```
{ tags = ...  
}
```

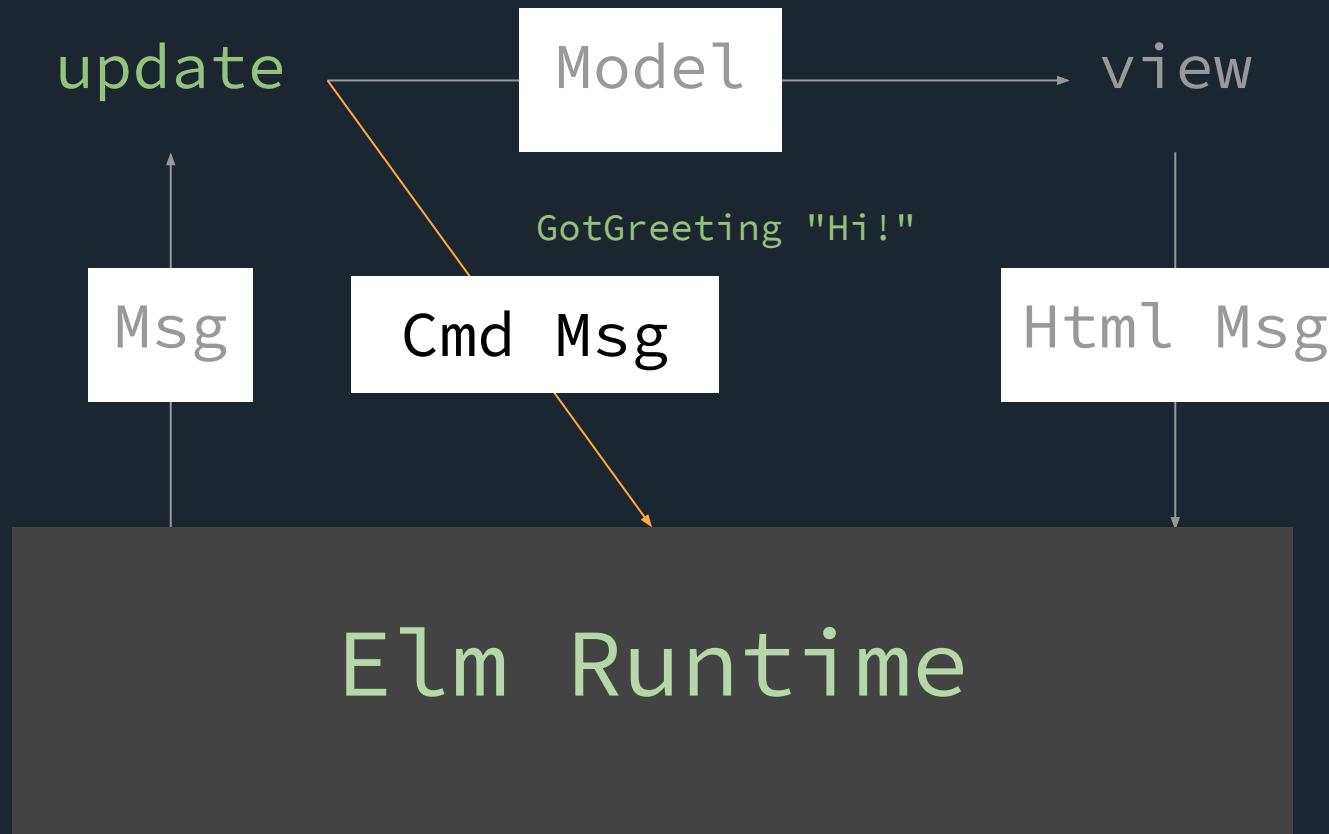




Elm Runtime



`pickGreeting : List String -> Cmd Msg`



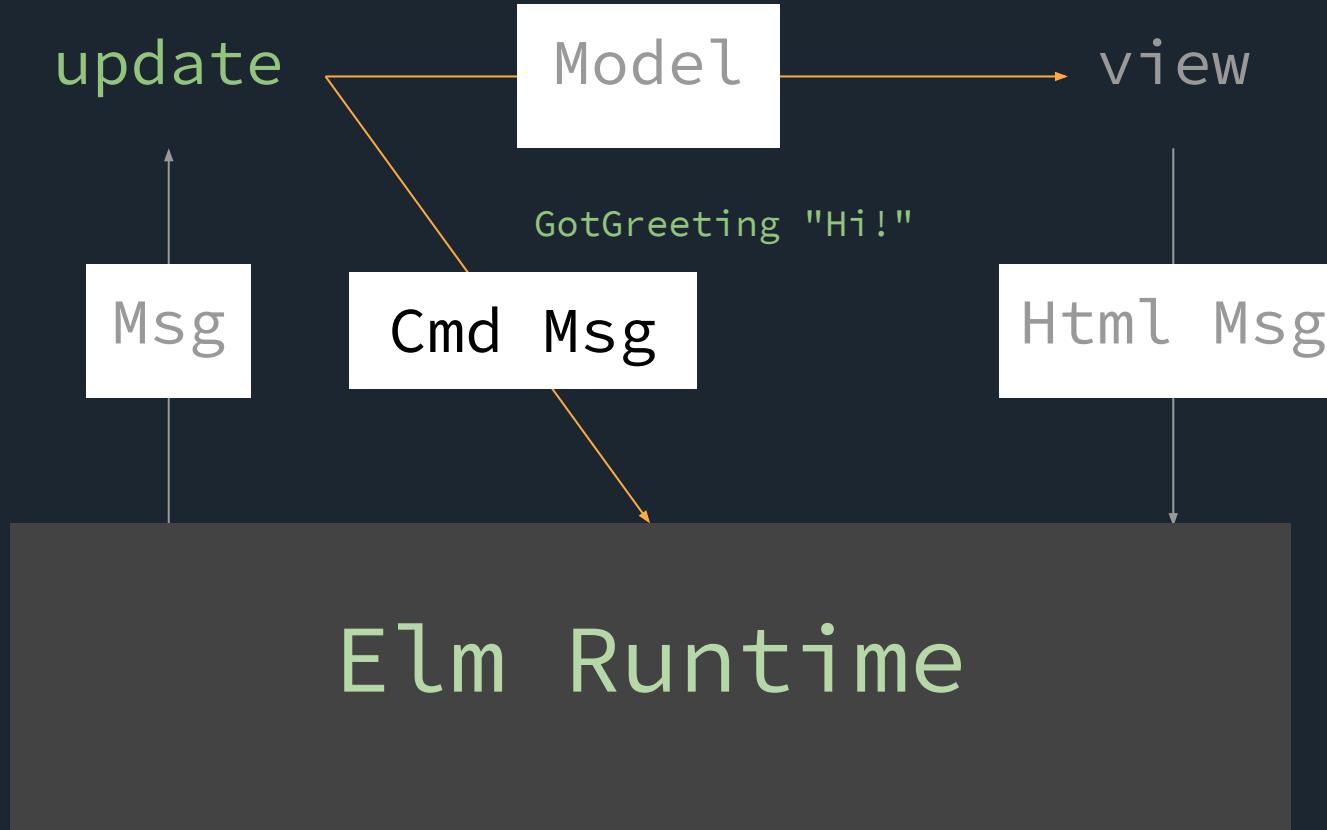
Browser.sandbox

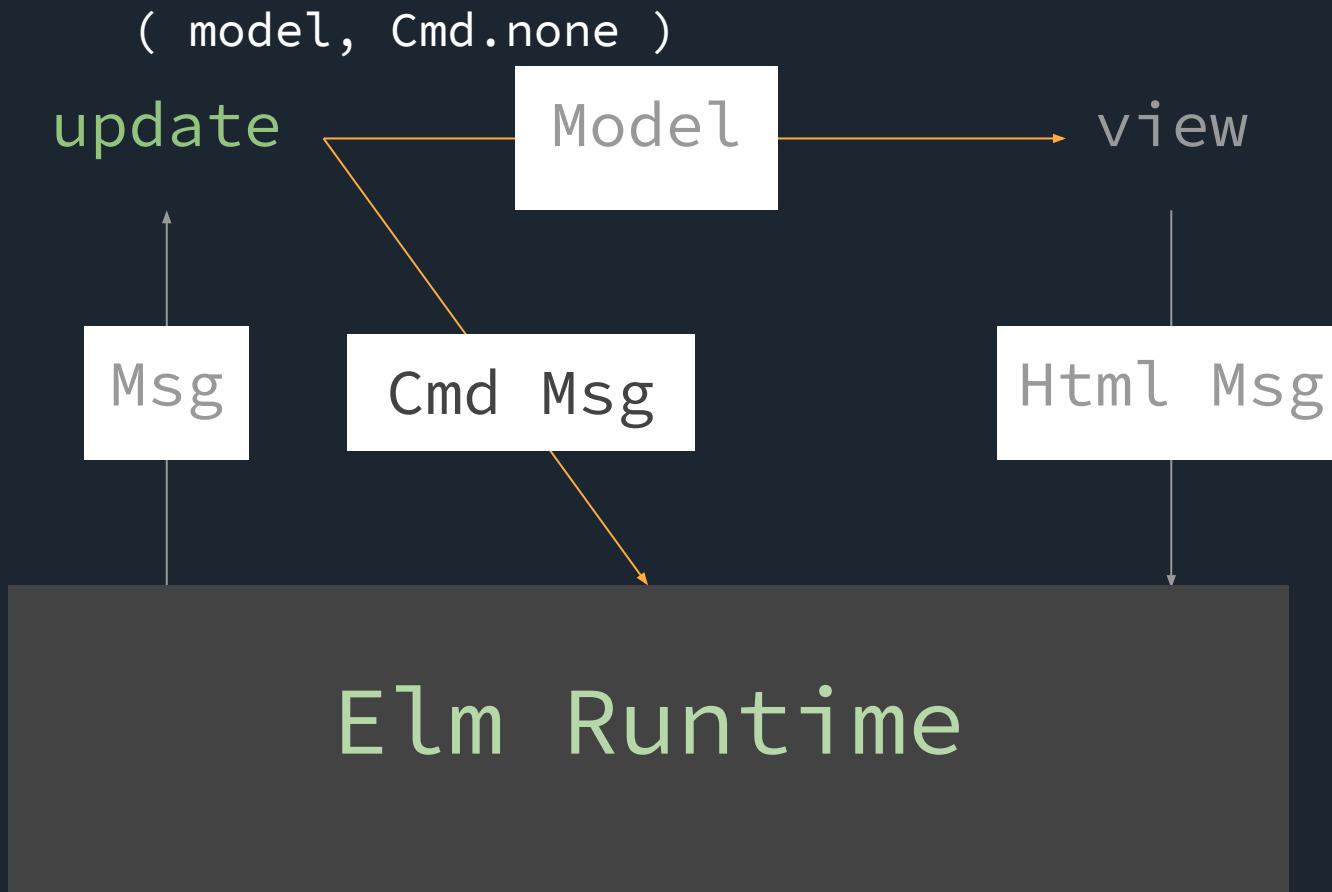
update : Msg -> Model -> Model

Browser.element

update : Msg -> Model -> (Model, Cmd Msg)

```
pickGreeting : List String -> Cmd Msg  
( model, pickGreeting greetings )
```





another function guarantee

no side effects

no modifying external state

fire-and-forget HTTP POST

fire-and-forget HTTP POST

✓ same arguments, same result

fire-and-forget HTTP POST

- ✓ same arguments, same result
- ✗ does not modify external state

side effects

~~side effects~~

managed effects

Cmd

Http.getString

what if it looked like this?

```
getString : String -> Cmd String
```

it **always** produces a string?

getString : String -> Request String

result if request succeeds

Http.send

Request → Cmd

translate failure into a **Msg**

translate success into a **Msg**

CompletedLoadFeed (Result [Error String])

result if request fails

```
Http.send  
  (\result -> CompletedLoadFeed result)  
  (Http.getString "/feed?tag=happiness")
```

Request → Cmd

translate failure into a **Msg**

translate success into a **Msg**

```
Http.send
```

```
CompletedLoadFeed
```

```
(Http.getString "/feed?tag=happiness")
```

Request → Cmd

translate failure into a **Msg**

translate success into a **Msg**

```
Http.getString "/feed?tag=happiness"  
|> Http.send CompletedLoadFeed
```

Request → Cmd

translate failure into a **Msg**

translate success into a **Msg**

```
cmd : Cmd Msg
cmd =
    Http.getString "/feed?tag=happiness"
        |> Http.send CompletedLoadFeed
```

Request → Cmd

translate failure into a **Msg**

translate success into a **Msg**

CompletedLoadFeed (Result Error (List Article))

-- success gives you a String

Http.getString url

-- success gives you a List Article

Http.get articlesDecoder url

case msg of

PATTERN MATCHING

CompletedLoadFeed (Ok articles) ->

...

CompletedLoadFeed (Err error) ->

...



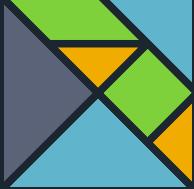
Review of Part 8

Tuples (Model, Cmd Msg)

Randomness Random.generate

Commands Description of an effect to be run

HTTP Http.send CompletedLoad request



Exercises for Part 8

Follow the instructions in [part8/README.md](#)

Complete the TODOs in **part8/Main.elm**



9. Talking to JavaScript

Subscriptions

Guarantees

Sending data to JS

Receiving data from JS

Responding to Custom Events

```
on : String -> Decoder msg -> Attribute msg
```

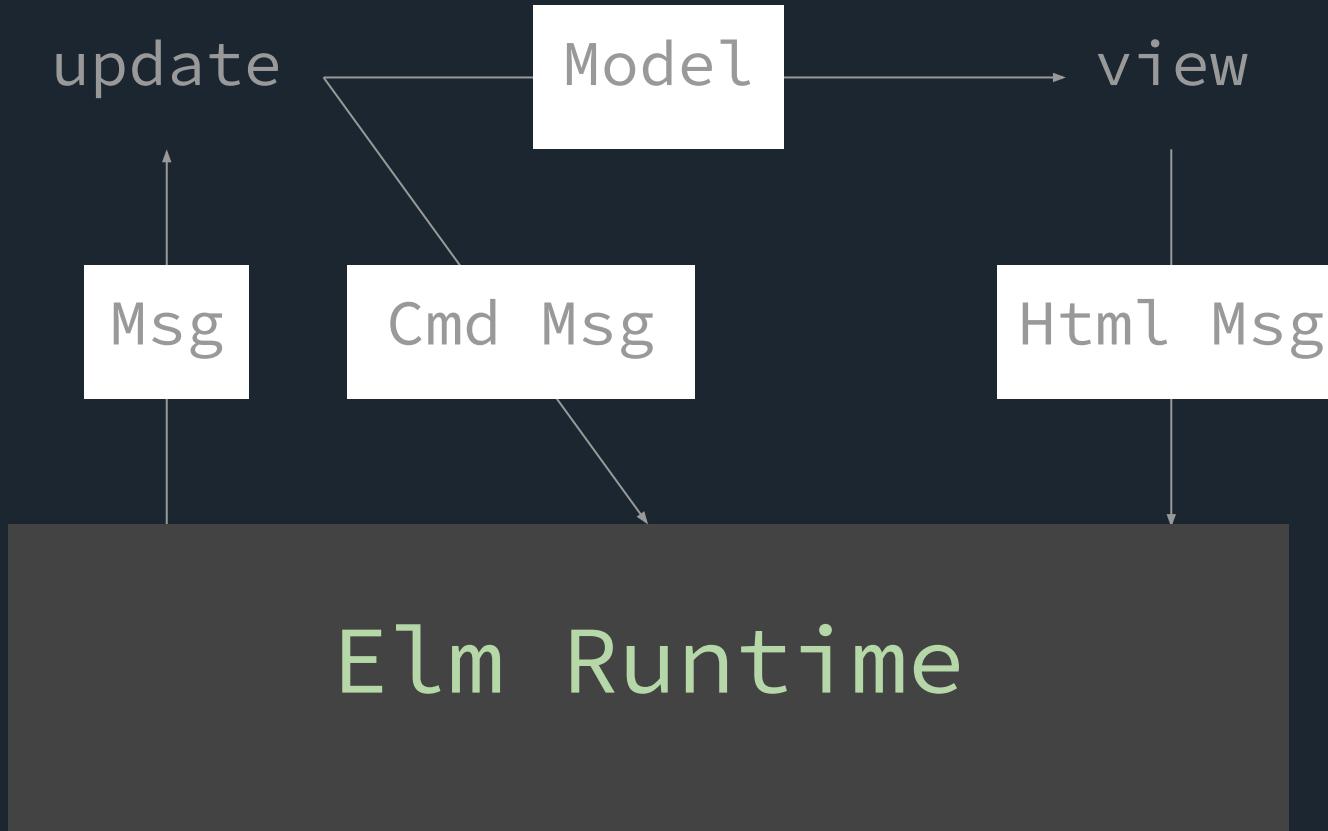
```
div [ on "mousemove" mousePointDecoder ] [...]
```

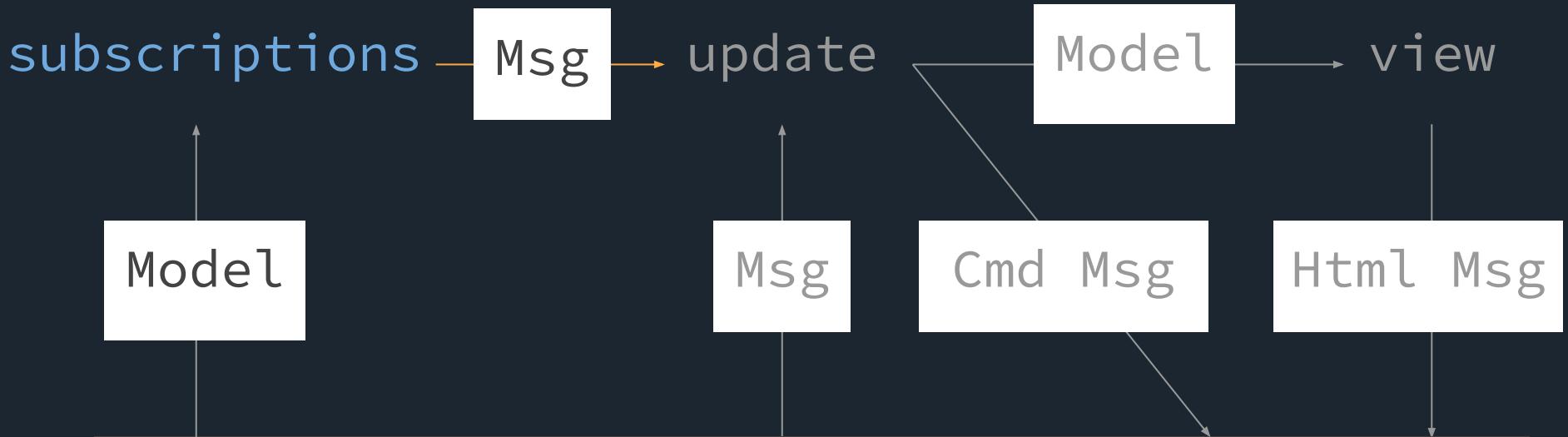
Responding to Window Events

on : String -> Decoder msg -> Attribute msg

Browser.onMouseMove : Decoder msg -> Sub msg

subscription





Elm Runtime

function guarantees

same arguments?
same return value

no side effects

all Elm functions are **pure**

Math.random()

```
localStorage.foo = "bar";
```

access huge JS ecosystem

while maintaining guarantees

"client/server" communication

Elm sends data to JS

JS sends data to Elm

no direct function calls!

Cmd on the Elm side



callback on JS side

Cmd Msg

~~Cmd~~ ~~Msg~~

Cmd msg

Type Variables

List.head : List elem -> Maybe elem

List.reverse : List val -> List val

List.length : List a -> Int

Unbound Type Variable

```
[] : List a
```

```
[ 1.1, 2.2 ] : List Float
```

```
[ "a", "b" ] : List String
```

Unbound Type Variable

[] : List a

String.concat : List String -> String

String.concat ["ab", "c"] → "abc"

String.concat [] → ""

Unbound Type Variable

`[] : List a`

`String.lines : String -> List String`

`String.lines "A\nB" → ["A", "B"]`

`String.lines "" → []`

Unbound Type Variable

```
[] : List a
```

```
div [ onClick Toggle ] [] : Html Msg
```

```
img [ src "logo.png" ] [] : Html a
```

```
img [ src "logo.png" ] [] : Html msg
```

"this is compatible with Html that produces Msg"

```
banner : Html Msg
banner =
    header []
        [ h1 [] [ text "conduit" ]
        , ...
        ]
```

"this is compatible with Html that produces anything"

```
banner : Html msg
banner =
    header []
        [ h1 [] [ text "conduit" ]
        , ...
        ]
```

```
banner : Html a
banner =
    header []
        [ h1 [] [ text "conduit" ]
        , ...
        ]
```

produces messages of type **Msg**
works with **update** functions that accept **Msg**

Cmd Msg

~~Cmd~~ ~~Msg~~

Cmd **msg**

works with any **update** function

...because it never produces any messages!

the **port** keyword

&

index.html

port module

```
port storeSession : Maybe String -> Cmd msg  
port onSessionChange : (String -> msg) -> Sub msg
```

Elm creates the implementation!

can be **null!**

```
port storeSession : Maybe String -> Cmd msg
```

```
port onSessionChange : (String -> msg) -> Sub msg
```

```
var app = Elm.Main.init({flags: session});
```

```
app.ports.storeSession.subscribe(function(str){  
    // JS code goes here  
});
```

```
app.ports.onSessionChange.send(someString);
```

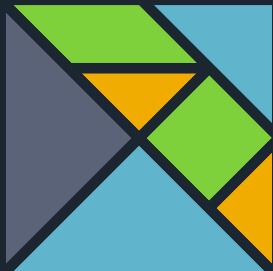
STORING SESSION DATA IN

localForage

```
localForage.getItem("key", function(val) { ... })
```

```
localForage.setItem("key", "val", function() { ... } )
```

Who owns this state?



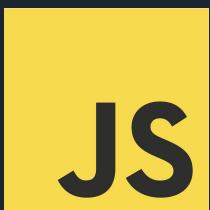
or



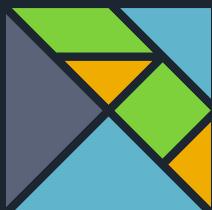
(pick one!)

STORING SESSION DATA IN

localForage



owns this state



caches it



Review of Part 9

Subscriptions

Sub `Msg`

Guarantees

all Elm functions are **pure**

Sending data to JS

`port send : Int -> Cmd msg`

Receiving data from JS

`port receive : ... -> Sub msg`



Exercises for Part 9

Follow the instructions in [part9/README.md](#)

Complete the TODOs in **part9/Main.elm**



Wrap-Up

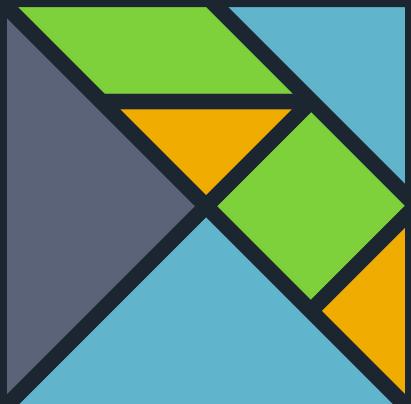
Summary

Resources

Community

Elm at Work

elm



JavaScript



compiles to



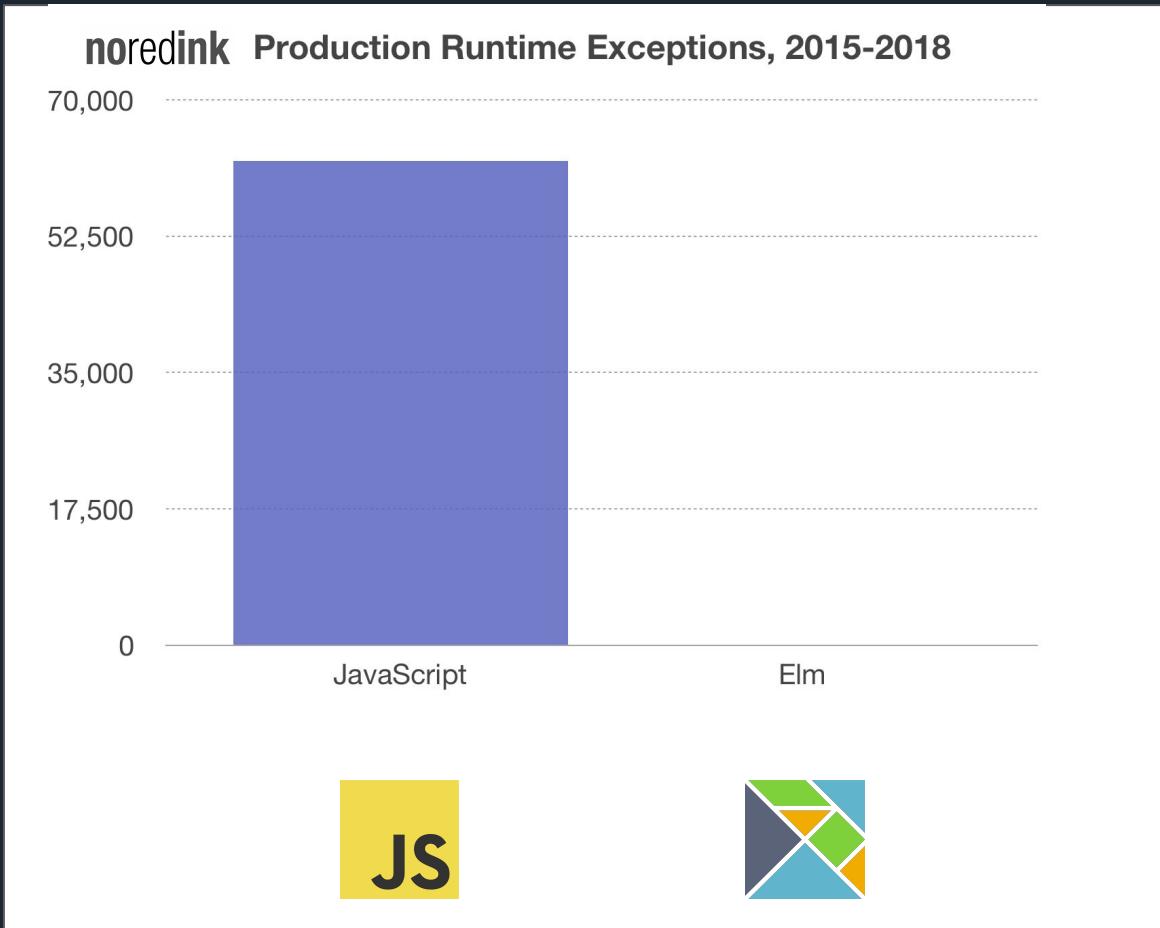
Companies of all sizes use Elm



FEATURE
SPACE

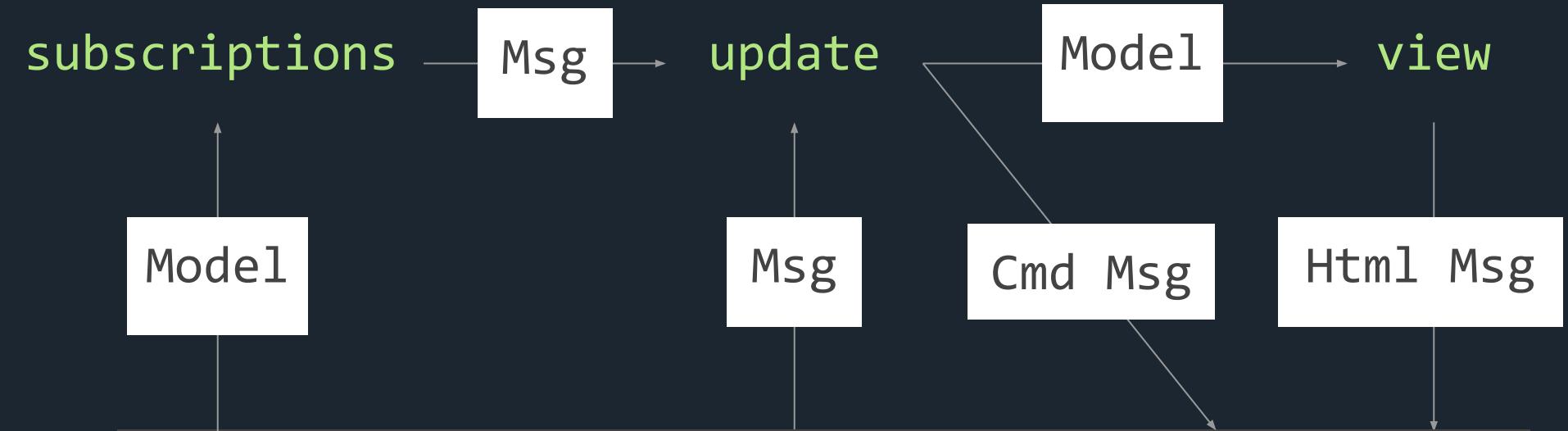


“No runtime exceptions”



```
<body>
  <div id="app"></div>
  <script>
    Elm.Main.init({
      node: document.getElementById("app")
    });
  </script>
</body>
```

INCREMENTAL ADOPTION



Elm Runtime

github.com/rtfeldman/elm-spa-example

conduit

Home Sign in Sign up

conduit

A place to share your knowledge.

Global Feed

 mrtsway
August 11, 2018

ertw
ertw



Popular Tags

- dragons
- training
- test
- tags
- as
- coffee
- flowers
- money
- happiness
- sushi
- caramel
- sugar
- japan
- ...
- ...

guide.elm-lang.org

Type to search

An Introduction to Elm

[Introduction](#)

[Install](#)

[Core Language](#)

[The Elm Architecture](#)

[User Input](#)

[Buttons](#)

[Text Fields](#)

[Forms](#)

[More](#)

[Effects](#)

An Introduction to Elm

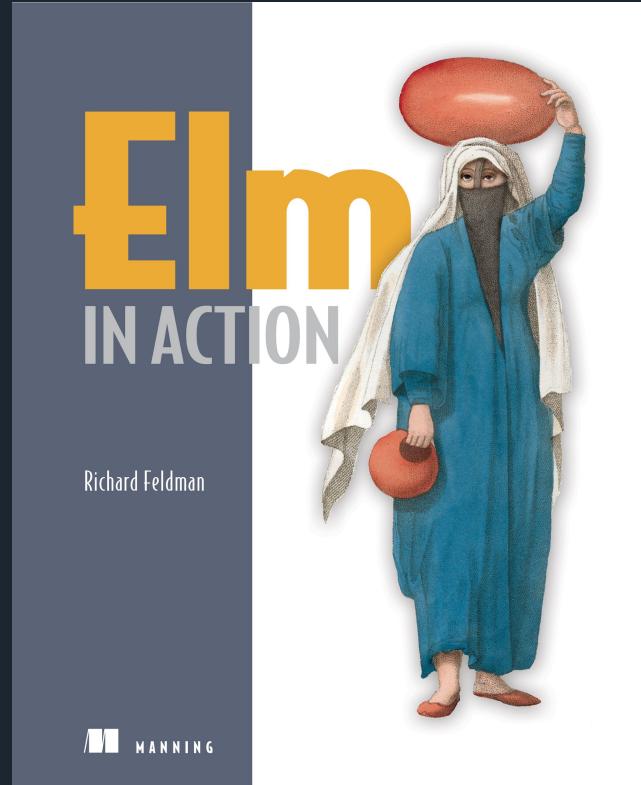
Elm is a functional language that compiles to JavaScript. It's a tool for creating websites and web apps. Elm has a lot of quality tooling.

This guide will:

- Teach you the fundamentals of programming in Elm
- Show you how to make interactive apps with Elm
- Emphasize the principles and patterns that get you started

By the end I hope you will not only be able to create Elm apps, but understand the core ideas and patterns that make Elm nice to use.

If you are on the fence, I can safely guarantee that once you learn Elm, you will end up writing better JavaScript and React apps.





elmlang.herokuapp.com

Elm ▾

rtfeldman

All Unreads

All Threads

Channels

beginners

Direct Messages

Apps

Gz

#beginners

☆ | 10,951 | 9 | FAQs at <http://faq.elm-community.org/>

Saturday, April 28th

Zukkari 7:16 AM
So how does one include external CSS in Elm? Do you have to set up gulp tasks and stuff? Or is inline style...

jessta 7:26 AM
@Zukkari the simplest thing to do it just include external css in the html file you embed Elm in.

Zukkari 7:26 AM
So this means manually modifying the compiled build? But what about development?

jessta 7:27 AM
a static css file included in the html doesn't involve any compiling

Zukkari 7:28 AM
Oh I get what you mean

jessta 7:28 AM
you just include the css and refer to those classes in your Elm views

Zukkari 7:28 AM
Yeah, thanks I will look into that

jessta 7:28 AM
Another option is to use elm-css, which lets you specify your CSS in Elm
<http://package.elm-lang.org/packages/rtfeldman/elm-css/latest>



discourse.elm-lang.org

 Elm

[all categories ▶](#) Latest [Top](#) [Categories](#)

Topic	Category	Users
<input checked="" type="checkbox"/> Best data structure for a collection of UI elements?	■ Learn	
Image annotation for the machine learning community	■ Show and Tell	
Understanding the Decoder type	■ Learn	
A Command Line Interface for Documentation	■ Show and Tell	

faq.elm-community.org



Elm (language) FAQ

Elm 0.16

Elm 0.17 Tasks

Elm 0.17

Elm FAQ

Elm FAQ

This page is mostly for folks *learning Elm*. It aggregates questions that are commonly asked on [the Slack channel](#), [the IRC channel](#), or [the mailing list](#). Those are all friendly and helpful places to go if you do not find the answer to your question here!

<https://elmtown.github.io>



*Elm
Town
Podcast*

SUCCESS STORIES

How Elm made our work better



Ossi Hanhinen
Hypertext Elementalist

TOPICS

functional, reactive, web, frontend, Elm

Christian Charukiewicz

HOME

Elm In Production: 25,000 Lines Later

One year of Elm in production



Tomas Zemanovic Mar 23 Originally published at tzemanovic.github.io on Mar 06, 2018

YOUR
SUCCESS
STORY
GOES
HERE!