Functional Reactive Programming in elm

H. Chase Stevens
chase@chasestevens.com

November 7, 2014

Abstract

Functional reactive programming has been proposed as a means of creating responsive, concurrent, and asynchronous web-based GUIs without necessitating inscrutable code. In this report, elm's implementation of functional reactive programming is described and evaluated. A toy search engine implementation is also presented in order to better demonstrate the key concepts of functional reactive programming as utilized in elm, as well as its relative advantages and disadvantages.

1 Introduction

When creating user interfaces, developers seek to keep all interactive elements of the interface responsive to user input, even when heavy computations are being performed or high-latency network calls are being made. Achieving this on the web typically has involved heavy use of callbacks, which induce unnecessary complexity and can make code structure difficult to reason about (Czaplicki, 2012).

Functional reactive programming is a paradigm intended to remedy this issue by offering as primitives asynchronous streams of user inputs which may be manipulated and used to drive program output and UI behaviour (Staltz, 2014). FRP also seeks to promote more intelligible code by imposing a functional, declarative approach, in contrast to traditional GUI development practices (Staltz, 2014).

The elm programming language incorporates the functional reactive programming paradigm, while also leveraging its principles to elimintate the need for directly specifying UI elements (Czaplicki and Chong, 2013). This report examines the functional reactive utilities supplied by elm, while also demonstrating their use via example. Further to this, a worked example of a simple search engine as implemented in elm is provided, with specific care taken to explain how functional reactive programming is used to process user input and return query results. Finally, the report concludes with an annotated list of resources available for further investigations into functional reactive programming.

2 Context

Typically, modern web interfaces make frequent network calls to remote servers to access data and performing computationally intensive tasks. Achieving responsive web interfaces is

therefore a matter of concurrently processing user input and updating interactive UI elements; a synchronous UI will otherwise become locked during a network call with high latency or over a low-bandwidth connection (Czaplicki, 2012).

In Javascript, the traditional solution to this problem is the use of callbacks, whereby asynchronous calls to network resources or computationally-intensive functions are supplied with a function to call upon their completion. Czaplicki (2012) contends that this style of programming leads to "responsive but unreadable" code, which necessarily has behaviour that is difficult to reason about, given that data will be passed from function to function in an unstructured, "spaghetti code"-like manner. Czaplicki (2012) further contends that maintaining such code has a high cognitive load, since the implicit callback structure will need to be investigated and analysed before changes to the code are made.

3 Functional reactive programming

In the functional reactive paradigm, asynchronous calls are abstracted away via use of "streams" (also referred to as "signals"). Streams are asynchronously updated series of events which may be created, manipulated, and combined in a myriad of ways (Staltz, 2014). Streams are used to represent not only user input, but also to wrap the program's output, resulting in a system which is responsive by default (Czaplicki, 2012). Streams may also represent data from remote servers, such as RSS feeds or API responses, as well as the results of internal program functions (Staltz, 2014). By ubiquitously utilizing the high-level stream concept and having streams of events directly drive program output and UI behaviour, Staltz (2014) notes that functional reactive programming can result in cleaner and more concise code that ultimately proves easier to reason about.

4 elm

4.1 Introduction

elm is a strongly-typed, Haskell-derived functional programming language which compiles into CSS, HTML, and Javascript (Czaplicki and Chong, 2013). In elm, functional reactive streams are incorporated as the `Signal` datatype. `Signal`s may be streams of arbitrary elements, for example, the `Mouse.position` stream is of type `Signal (Int,Int)` (elm, 2013a). In this way, `Signal`s bear some resemblance to monads as implemented in Haskell.

4.2 `Signal` manipulation in elm

elm provides several notable functions for transforming `Signal`s, as documented in the standard library catalog (elm, 2013b):

4.2.1 `constant`

`constant`, with type `a -> Signal a`, takes a value and creates a constant, unchanging `Signal` from it. This is particularly useful when displaying static text, e.g.

```
main = constant (plainText "Hello world")
```

4.2.2 lift

`lift`, with type `(a -> b) -> Signal a -> Signal b`, allows a function to be applied to the events of a `Signal`. This can been viewed as an equivalent to Haskell's `fmap` function, which operates on Monads. As an example:

```
main = constant 1
       |> lift ((*) 2)
       |> lift asText
```

transforms the constant stream

```
- 1 - 1 - 1 - 1 - 1 -
```

into

```
- 2 - 2 - 2 - 2 - 2 -
```

and then, finally, into

```
- Element "2" - Element "2" - Element "2" - Element "2" - Element "2" -
```

Related functions, `lift2` through `lift8`, take functions taking between two and eight arguments as well as the appropriate number of `Streams`.

4.2.3 merge

`merge`, with type `Signal a -> Signal a -> Signal a`, combines two streams into a single stream, dropping events from the second stream if there is a simultaneous event in the first stream. As an example: supposing one wanted to create a Tetris game in elm, which updated the game state either every two seconds (allowing blocks to fall) or whenever the user pressed a key. Using `merge`, with `['U','P','D','A','T','E']` serving as an indicator to cause blocks to fall every two seconds, one might write:

```
main = Keyboard.keysDown
       |> lift (map Char.fromCode)
       |> flip merge (constant ['U','P','D','A','T','E']
                      |> sampleOn (every (2 * second)))
       |> lift updateGame
```

4.2.4 foldp

`foldp`, with type `(a -> b -> b) -> b -> Signal a -> Signal b`, accumulates values from a stream given an accumulator function and a base case (`b`). In this way, it can be considered similar to Haskell's `foldl`. As an example, to display all key presses made by the user:

```
main = Keyboard.keysDown
       |> lift (map Char.fromCode)
       |> foldp (flip (++)) []
       |> lift (String.fromList >> asText)
```

An interesting special case of `foldp` is `count`, which can be defined thusly:

```
count = foldp (\x y -> y + 1) 0
```

Regardless of their type, `count` will increment on every event from the given stream.

4.2.5 `keepIf`

`keepif`, with type `(a -> Bool) -> a -> Signal a -> Signal a`, allows filtering events from a stream given a predicate and some base case (`a`). For example,

```
main = Mouse.clicks
       |> count
       |> keepIf ((>) 5) 0
       |> lift asText
```

will display the number of mouse clicks up to a maximum of 4.

4.2.6 `sampleOn`

Finally, `sampleOn`, with type `Signal a -> Signal b -> Signal b`, produces a stream of events from `Signal b` which updates only when `signal a` does. This is especially useful in combination with builtin functions like `every second`, as shown in the example in 4.2.3.

5 Example

5.1 Overview

The example created for this report was a simple, one-page search engine. The engine employs the tf-idf algorithm to allow users to search for tourist destinations within Scotland. The complete code for this example is given in the appendix on page 10. The code was loosely adapted from the example given at `http://elm-lang.org/edit/examples/Reactive/TextField.elm`.

5.2 UI elements

At the top of the search engine page, a text input is displayed, into which the user types their query. Below this, the query results (which consist of a title, a description, and, to the right, a photograph) are displayed in descending order of relevance. These results are updated dynamically in near-real-time, as the user types their query.

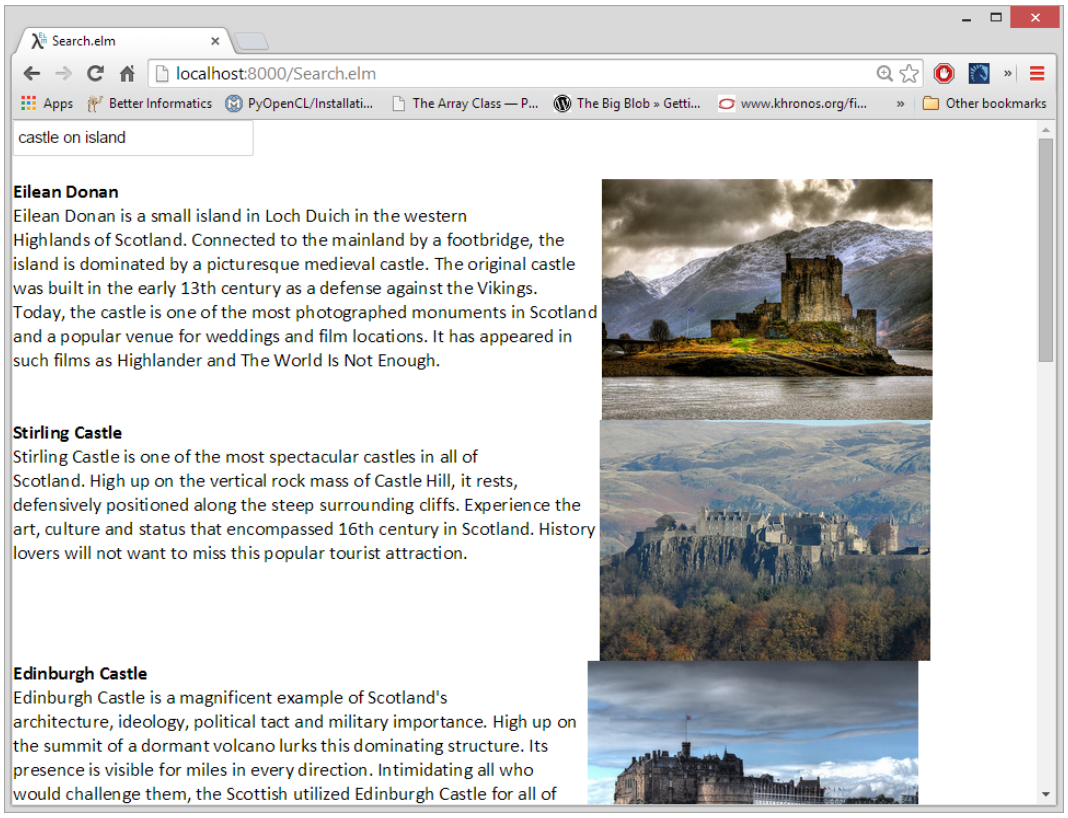

Figure 1: Example search engine running using `elm-reactor`.

5.3 Implementation

5.3.1 Input signal

The search engine is driven from a single input signal, namely, a `Graphics.Input.Field` text field input signal:

```
import Graphics.Input (Input, input)
import Graphics.Input.Field as Field

searchBar : Input Field.Content
searchBar = input Field.noContent
```

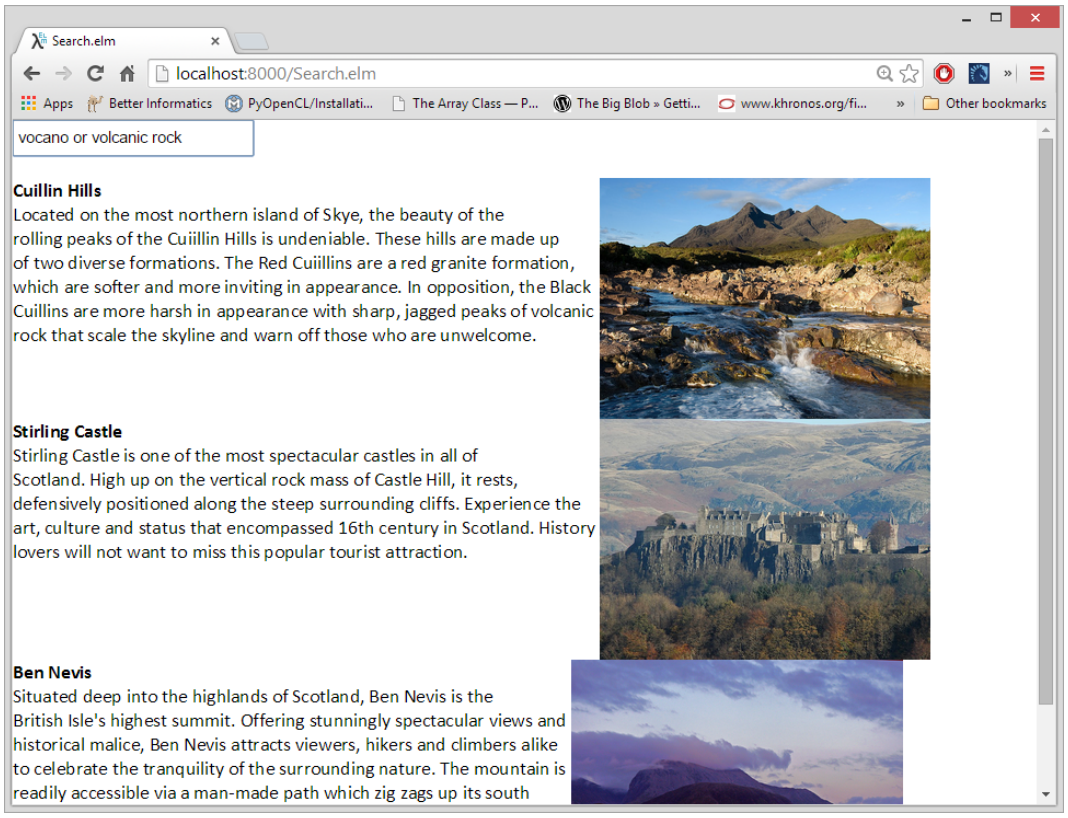

Figure 2: A second user query, and its results.

5.3.2 `main`

In `main`, the above signal is used as input to both a function which yields a signal for the search bar element at the top of the page (`searchBox`, `lifted`) and a function which yields a signal for the result element displayed below (`scene`, also `lifted`). A third, constant newline signal is used to divide the two elements. These three signals are `combined` and laid out by using `lift (flow down)`, resulting in the `Signal Element` return value.

```
main : Signal Element
main = let search = Field.field Field.defaultStyle searchBar.handle identity
  in
    combine
    [ lift (search "Enter search") searchBar.signal
    , plainText "\n" |> constant
    , dropRepeats searchBar.signal
      |> sampleOn (every (2 * second))
      |> lift scene
    ] |> lift (flow down)
```

5.3.3 scene

The `scene` function takes `searchBar`'s contents as input. Its tokenized contents are passed into `attractionsByRelevance`, which returns a list of attractions (`records`) sorted by query-relevance (descending). Each of these is then converted into a "result" element (containing title, description, and image) by the inline `result` function. Finally, the individual results are combined into a single element via `flow down`.

```
scene : Field.Content -> Element
scene fieldContent =
  let sortedAttractions = tokens fieldContent.string
                          |> attractionsByRelevance
      result attraction = [ (toText >> bold) attraction.title
                          , toText attraction.body, toText "\n"
                          ]
                          |> map leftAligned
                          |> flow down
                          |> flip (::) [image 275 200 attraction.imageUrl]
                          |> flow right
  in
    map result sortedAttractions |> flow down
```

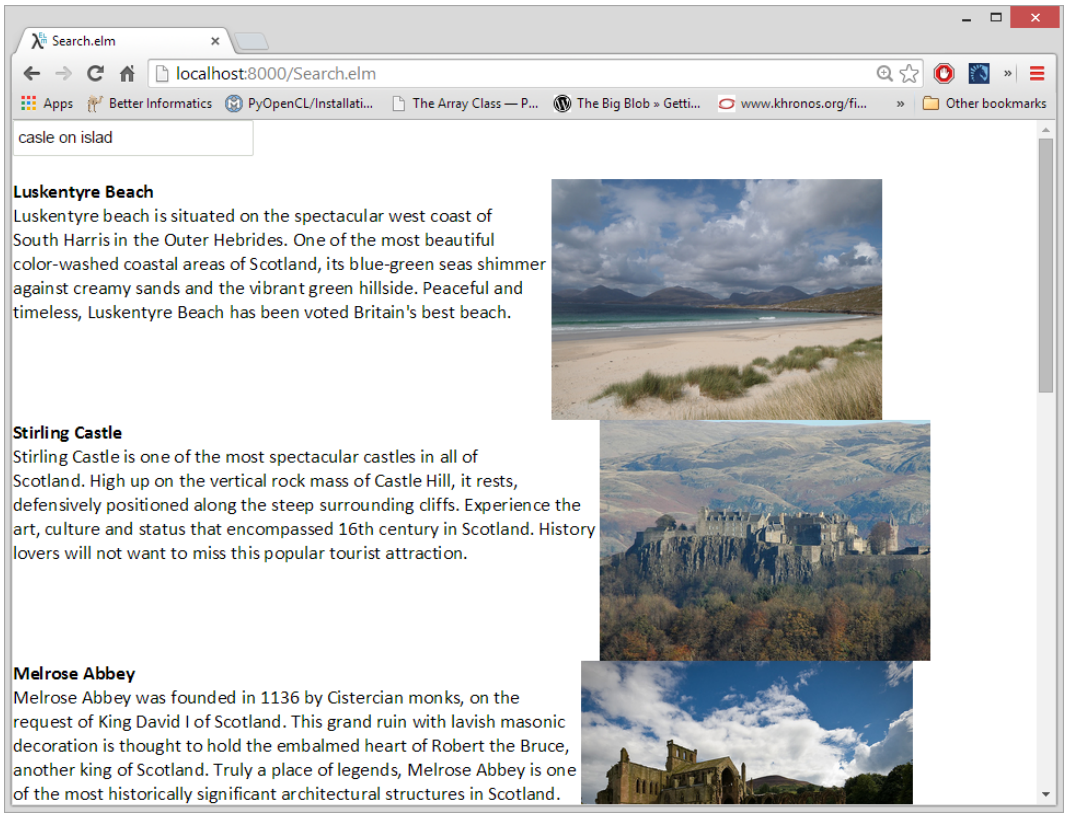

Figure 3: User query typed into the text input element is partially discarded.

5.4 Development

Because ranking the attractions by tf-idf score is a nontrivial computation, naively passing the text field's signal directly to `scene` within `main` (as below) results in a text field that is not only unresponsive, but, worse, loses user input (see Figure 3).

```
let finalElements fieldContent =
  [ search "Enter search" fieldContent
  , scene fieldContent  -- results in unresponsive text field
  ] |> flow down
in
  lift finalElements searchBar.signal
```

To combat this, the user input signal is instead first filtered down (via `dropRepeats`) into a signal which contains only events which reflect a change in the input, e.g. from

```
- A - A - A - B - B - C - D - D -
```

to

```
- A - - - - - B - - - C - D - - -
```

Further to this, the resultant stream is sampled only once every two seconds - unfortunately, shorter time periods still cause issues with loss of user input. This transformation can be visualized as follows:

```
- A - - - - - B - - - C - D - - -
```

to (assuming five dashes are equivalent to two seconds):

```
- - - - A - - - - B - - - - D - -
```

The final process is therefore:

```
combine
[ lift (search "Enter search") searchBar.signal
, dropRepeats searchBar.signal  -- keep only changes
  |> sampleOn (every (2 * second))  -- events every two seconds
  |> lift scene  -- pass to scene, which ranks attractions
] |> lift (flow down)
```

Note that this requires "splitting" `searchBar.signal` (as mentioned in 5.3.2), as otherwise the search engine's text input field would only accept keystrokes every two seconds. The two results of type `Signal Element` are then put into a list (`[Signal Element]`), which is transformed into type `Signal [Element]` by `combine`.

Were the relevancy rankings a server-side computation requested via AJAX, the resultant signal would have been asynchronous by default, which would allow for displaying the ranked results as soon as they are ready. In Asynchronous Functional Reactive Programming for GUIs, Czaplicki and Chong (2013) describe an `async` primitive for elm which allows any signal to be marked for asynchronous computation, allowing these signals to be arbitrarily computationally expensive without compromising GUI responsiveness. This, however, does not appear in the current implementation of elm.

6 Resources

6.1 The introduction to Reactive Programming you've been missing

In this tutorial, Staltz provides an excellent overview of functional programming concepts, as well as supplying motivation for the paradigm's usage. Multitudinous examples are provided, with illustrations to help visualize how various manipulations affect stream output and behaviour. Staltz then offers an example of a Twitter-esque "Who to follow" interface element as implemented through the RxJS functional reactive Javascript library, helpfully explaining design and implementation decisions along the way.

6.2 Controlling Time and Space: understanding the many formulations of FRP

In this 40-minute presentation, elm language creator Czaplicki gives a brief introduction to elm and the functional reactive programming paradigm, with special note made of elm's "time-travelling" debugger and hot-swap ability. More notably, Czaplicki explicates the design decisions behind elm's synchronous-by-default flavor of functional reactive programming, and contrasts this approach to other systems in which stream graphs can be reconfigured dynamically (higher-order functional reactive programming), systems in which streams are asynchronous by default (ansynchronous data flow), and other applications of streams outside of interfaces (arrowized functional reactive programming).

7 Conclusion

While still in its (relative) infancy, elm offers a well-developed and surprisingly intuitive means by which to design and develop user interfaces via functional reactive programming. Furthermore, by employing functional reactive programming to allow the automated generation of both graphical elements and their computational "backend", elm makes a strong case for the use of functional reactive languages and libraries over traditional callback-based asynchronous approaches. However, without mature support for asynchronous streams outside of AJAX requests, creating truly responsive systems with elm can be a challenge.

References

E. Czaplicki. (2012) Escape from callback hell: Callbacks are the modern goto. Accessed 6th November 2014. [Online]. Available: http://elm-lang.org/learn/Escape-from-Callback-Hell.elm

A. Staltz. (2014) The introduction to reactive programming you've been missing. Accessed 6th November 2014. [Online]. Available: https://gist.github.com/staltz/868e7e9bc2a7b8c1f754

E. Czaplicki and S. Chong, "Asynchronous functional reactive programming for guis," in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design*

and Implementation (PLDI), June 2013, pp. 411–422, accessed 6 November 2014. [Online]. Available: http://people.seas.harvard.edu/~chong/pubs/pldi13-elm.pdf

(2013) Mouse. Elm standard library documentation, accessed 6th November 2014. [Online]. Available: http://library.elm-lang.org/catalog/elm-lang-Elm/0.13/Mouse

(2013) Signal. Elm standard library documentation, accessed 6th November 2014. [Online]. Available: http://library.elm-lang.org/catalog/elm-lang-Elm/0.13/Signal

E. Czaplicki. (2014, September) Controlling time and space: understanding the many formulations of frp. Strange Loop. Accessed 6th November 2014. [Online]. Available: https://www.youtube.com/watch?v=Agu6jipKfYw

Appendices

Appendix A: Example source code

```
import String
import Char
import Set
import List
import Dict
import Text
import Graphics.Input (Input, input)
import Graphics.Input.Field as Field

-- based on the example given at
-- http://elm-lang.org/edit/examples/Reactive/TextField.elm
-- (Accessed 4th November 2014):
searchBar : Input Field.Content
searchBar = input Field.noContent

main : Signal Element
main = let search = Field.field Field.defaultStyle searchBar.handle identity
  in
    combine
    [ lift (search "Enter search") searchBar.signal
    , plainText "\n" |> constant
    , dropRepeats searchBar.signal
      |> sampleOn (every (2 * second))
      |> lift scene
    ] |> lift (flow down)

scene : Field.Content -> Element
```

```
scene fieldContent =
  let sortedAttractions = tokens fieldContent.string
                          |> attractionsByRelevance
      result attraction = [ (toText >> bold) attraction.title
                          , toText attraction.body, toText "\n"
                          ]
                          |> map leftAligned
                          |> flow down
                          |> flip (::) [image 275 200 attraction.imageUrl]
                          |> flow right
  in
    map result sortedAttractions |> flow down

type Attraction = { title: String, body: String, imageUrl: String }

-- taken from
-- http://www.touropia.com/tourist-attractions-in-scotland/
-- (Accessed 4th November 2014):
attractions : [Attraction]
attractions =
  [
    {title = "Broch of Mousa",
     body = "One of the most prestigious and well-preserved brochs in the
Shetland Islands, this impressive structure is a rotund tower lined with
stone internally and externally to provide the optimum strength as a
defensive structure. The tower was built around 100 BC and is the only
broch which is complete right to the top, including the original
intramural stair.",
     imageUrl =
      "http://farm2.static.flickr.com/1160/1427611613_a686a1854d_z.jpg?zz=1"
     }
  , {title = "Melrose Abbey",
     body = "Melrose Abbey was founded in 1136 by Cistercian monks, on the
request of King David I of Scotland. This grand ruin with lavish masonic
decoration is thought to hold the embalmed heart of Robert the Bruce,
another king of Scotland. Truly a place of legends, Melrose Abbey is one
of the most historically significant architectural structures in Scotland.",
     imageUrl =
      "http://farm4.static.flickr.com/3153/2729634256_863d2b2176_z.jpg?zz=1"
     }
  , {title = "Cuillin Hills",
     body = "Located on the most northern island of Skye, the beauty of the
rolling peaks of the Cuiillin Hills is undeniable. These hills are made up
of two diverse formations. The Red Cuiillins are a red granite formation,
which are softer and more inviting in appearance. In opposition, the Black
```

```
Cuillins are more harsh in appearance with sharp, jagged peaks of volcanic
rock that scale the skyline and warn off those who are unwelcome.",
     imageUrl =
      "http://farm3.static.flickr.com/2448/3633666760_f947b66942_z.jpg?zz=1"
     }
  , {title = "Skara Brae",
     body = "Located on the main island of Orkney, Skara Brae is one of the
best preserved Stone Age villages in Europe. It was covered for hundreds
of years by a sand dune until a great storm exposed the site in 1850. The
stone walls are relatively well preserved because the dwellings were
filled by sand almost immediately after the site was abandoned. Older than
Stonehenge and the Great Pyramids, it has been called the \"Scottish
Pompeii\" because of its excellent preservation.",
     imageUrl =
      "http://farm1.static.flickr.com/96/251621816_f4d2c13d90_z.jpg?zz=1"
     }
  , {title = "Stirling Castle",
     body = "Stirling Castle is one of the most spectacular castles in all of
Scotland. High up on the vertical rock mass of Castle Hill, it rests,
defensively positioned along the steep surrounding cliffs. Experience the
art, culture and status that encompassed 16th century in Scotland. History
lovers will not want to miss this popular tourist attraction.",
     imageUrl =
      "http://farm1.static.flickr.com/183/483655010_d9c9794fbd_z.jpg?zz=1"
     }
  , {title = "Luskentyre Beach",
     body = "Luskentyre beach is situated on the spectacular west coast of
South Harris in the Outer Hebrides. One of the most beautiful
color-washed coastal areas of Scotland, its blue-green seas shimmer
against creamy sands and the vibrant green hillside. Peaceful and
timeless, Luskentyre Beach has been voted Britain's best beach.",
     imageUrl =
      "http://farm6.static.flickr.com/5191/5902560358_914edfa75e_z.jpg?zz=1"
     }
  , {title = "Loch Ness",
     body = "One of the most famous lakes in the world, Loch Ness is the second
largest loch in Scotland after Loch Lomond (and due to its great depth it
is the largest by volume). About a mile wide at most places it holds the
legend of an infamous sea monster. The most notorious mythical creature of
modern time, Nessie, is said to dwell in the lake. With an air of mystery,
the intriguing area of Loch Ness should not be missed. You might even get
a glimpse of Nessie!",
     imageUrl =
      "http://farm3.static.flickr.com/2712/4441473182_a5125fec3e_z.jpg?zz=1"
     }
```

```
  , {title = "Ben Nevis",
     body = "Situated deep into the highlands of Scotland, Ben Nevis is the
British Isle's highest summit. Offering stunningly spectacular views and
historical malice, Ben Nevis attracts viewers, hikers and climbers alike
to celebrate the tranquility of the surrounding nature. The mountain is
readily accessible via a man-made path which zig zags up its south
westerly face, while the rock face on the north west of the mountain is
strictly for experienced mountaineers only.",
     imageUrl =
      "http://farm5.static.flickr.com/4154/4963323610_47f0ebdef4_z.jpg?zz=1"
     }
  , {title = "Eilean Donan",
     body = "Eilean Donan is a small island in Loch Duich in the western
Highlands of Scotland. Connected to the mainland by a footbridge, the
island is dominated by a picturesque medieval castle. The original castle
was built in the early 13th century as a defense against the Vikings.
Today, the castle is one of the most photographed monuments in Scotland
and a popular venue for weddings and film locations. It has appeared in
such films as Highlander and The World Is Not Enough.",
     imageUrl =
      "https://c2.staticflickr.com/4/3293/2433869373_15e31d33d2_z.jpg?zz=1"
     }
  , {title = "Edinburgh Castle",
     body = "Edinburgh Castle is a magnificent example of Scotland's
architecture, ideology, political tact and military importance. High up on
the summit of a dormant volcano lurks this dominating structure. Its
presence is visible for miles in every direction. Intimidating all who
would challenge them, the Scottish utilized Edinburgh Castle for all of
their major battles and military strategizing. A strong standing symbol
of their perseverance and struggle for independence, Edinburgh Castle is
one of the top tourist attractions in Scotland.",
     imageUrl =
      "https://c4.staticflickr.com/8/7386/9550271050_32bf1bf589_c.jpg?zz=1"
     }
  ]

alphanumericCharacters = String.toList "abcdefghijklmnopqrstuvwxyz0123456789"
                         |> Set.fromList

startingLetters : String -> String
startingLetters string = let isAlnum = flip Set.member alphanumericCharacters
  in
    String.toList string
    |> List.partition isAlnum
    |> fst >> String.fromList
```

```
tokens : String -> [String]
tokens string = String.words string
                |> map String.toLower
                |> map startingLetters

unique : [String] -> [String]
unique xs = Set.fromList xs |> Set.toList

counts : [String] -> Dict.Dict String Int
counts tokens = let uniqueTokens = unique tokens
                    count item list = List.filter ((==) item) list
                                      |> List.length
                    tokenCount = flip count tokens
  in
    map tokenCount uniqueTokens
    |> zip uniqueTokens
    |> Dict.fromList

-- tf-idf algorithm adapted from personal previous coursework for
-- Text Technologies for Data Science (INFR11100)
-- Assessment 1, 6th October 2014
idf : String -> Float
idf token = let documents = map (.body >> tokens >> Set.fromList) attractions
                tokenFrequency = filter (Set.member token) documents
                                 |> List.length
  in
     toFloat (List.length attractions) / toFloat tokenFrequency |> logBase 2
```

```
relevance : [String] -> [String] -> Float
relevance query document =
  let mutualTokens = (Set.fromList query, Set.fromList document)
                     |> uncurry Set.intersect
                     |> Set.toList
      queryCounts = counts query
      queryCount token = Dict.getOrElse 0 token queryCounts |> toFloat
      documentCount token = Dict.getOrElse 0 token documentCounts |> toFloat
      documentCounts = counts document
      k = 2
      documents = map (.body >> tokens) attractions
      averageK = (sum >> toFloat) (map length documents)
                 / (length >> toFloat) documents
      documentSquash token = documentCount token
                             + (averageK * toFloat (length document))
      tfidf token = queryCount token * documentCount token
                    / documentSquash token * idf token
  in
    map tfidf mutualTokens |> sum

attractionsByRelevance : [String] -> [Attraction]
attractionsByRelevance query = attractions
                               |> map (.body >> tokens >> relevance query)
                               |> flip zip attractions
                               |> filter (fst >> (<) 0)
                               |> List.sortBy fst
                               |> List.reverse
                               |> map snd
```