Monads in Javascript

Jana Karcheska

Netcetera, Skopje

What makes apps messy?

- Impurity
  - Nulls
  - Callbacks
  - Errors
  - Side effects

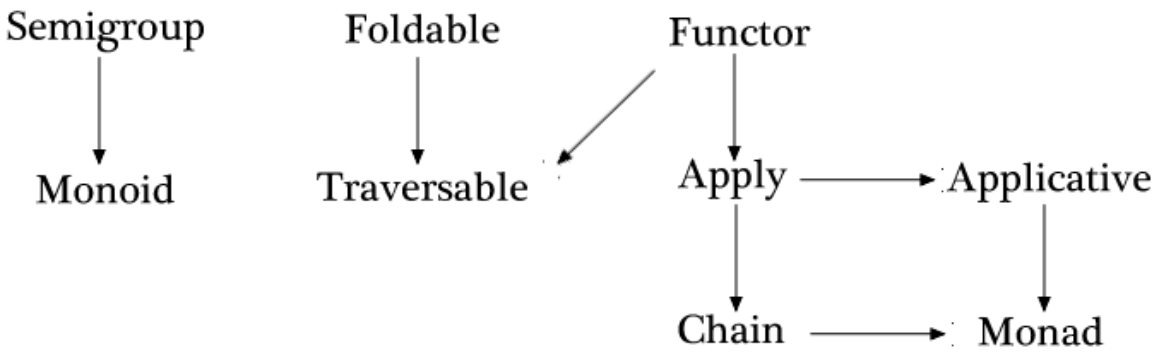
Stay pure

- Separate the pure from the impure
  - Date, random
  - DOM manipulation
  - Handling user input

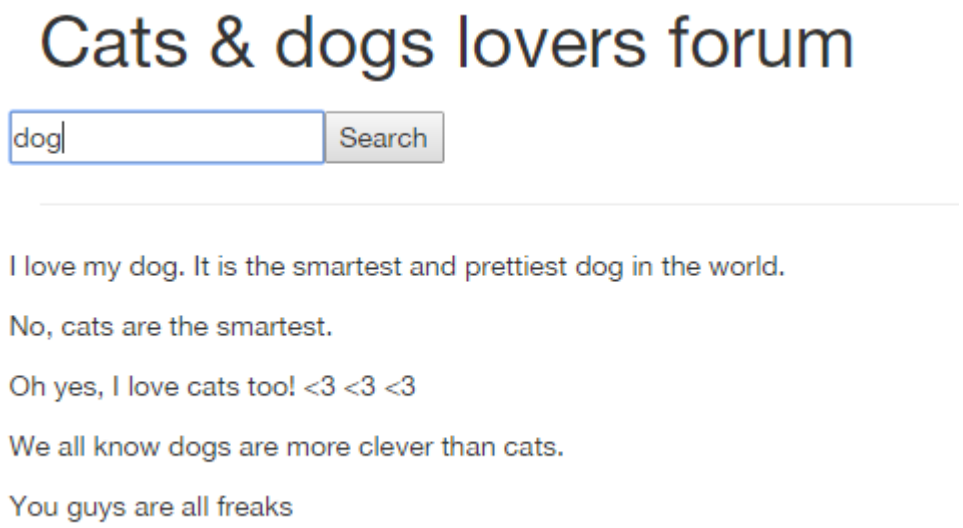
What are monads?

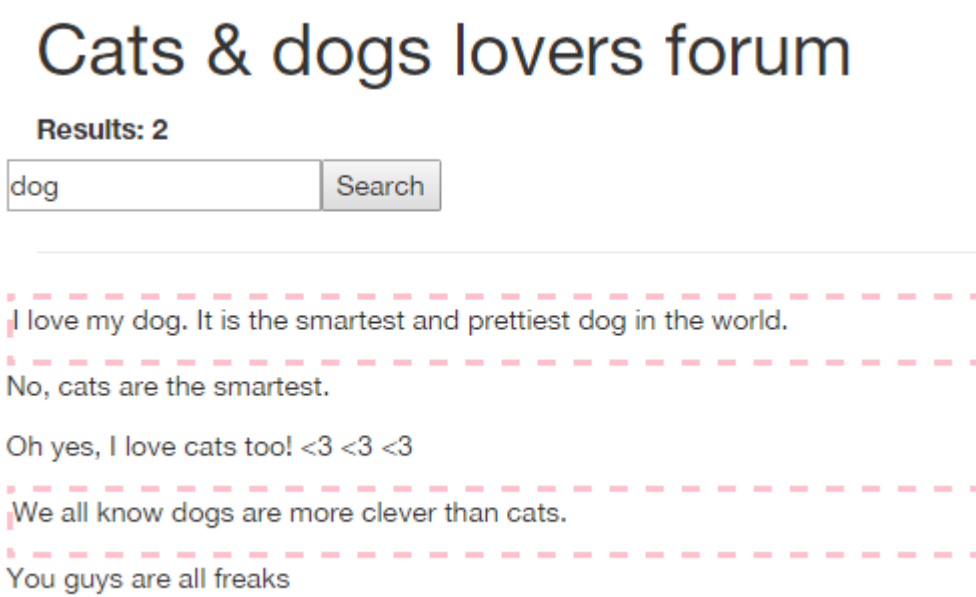
- Category theory
  - You don’t need to know category theory to use monads

- Functional programming
  - Haskell – loophole to introduce the illusion of impurity



Today’s example





Our goal for today

```
function search() {
  var searchWord = $("#searchWord").val();
  var allCommentElements = getAllCommentElements();
  var resultsNumber = 0;

  allCommentElements.each(function (index, el) {
    if (matchText($(el).text(), searchWord)) {
      applyResultStyle($(el).parent());
      resultsNumber++;
    }
  });

  var resultsNumberElement = $('#resultsNumber');
  resultsNumberElement.text('Results: ' + resultsNumber);
}
```

```
// get the search word from the search input field
function getSearchWord(elName) {
  return $(elName).val();
}
// find all comment p elements
function getAllCommentElements() {
  return $("p").toArray();
}
// get comment text from p element
function getCommentFromElement(el) {
  return $(el).text();
}
// text matching
function matchText(searchWord, comment) {
  return comment.split(searchWord).length - 1;
}
// apply style to mark search results
function applyResultStyle(element) {
  $(element).parent().css({'border-style': 'dashed', 'border-color' : 'pink'});
}
```

```
var matchElement = function(searchWord) {
  return compose(_.curry(matchText)(searchWord), getCommentFromElement);
}

var doSearch = function(searchWord) {
  return _.filter(matchElement(searchWord))(getAllCommentElements());
}

var getResults = compose(mjoin, map(_.forEach(applyResultStyle)), map(doSearch), Maybe, getSearchWord);

function search() {
  var results = getResults("#searchWord")
  var resultsNumberElement = $('#resultsNumber');
  resultsNumberElement.text('Results: ' + results.length);
}
```

```
// get the search word from the search input field
function getSearchWord(elName) {
  return $(elName).val();
}
// find all comment p elements
function getAllCommentElements() {
  return $("p").toArray();
}
// get comment text from p element
function getCommentFromElement(el) {
  return $(el).text();
}
// text matching
function matchText(searchWord, comment) {
  return comment.split(searchWord).length - 1;
}
// apply style to mark search results
function applyResultStyle(element) {
  $(element).parent().css({'border-style': 'dashed', 'border-color' : 'pink'});
}
```

Example – imperative programming

```
function search() {
  var searchWord = $("#searchWord").val();
  var allCommentElements = getAllCommentElements();
  var resultsNumber = 0;

  allCommentElements.each(function (index, el) {
    if (matchText($(el).text(), searchWord)) {
      applyResultStyle($(el).parent());
      resultsNumber++;
    }
  });

  var resultsNumberElement = $('#resultsNumber');
  resultsNumberElement.text('Results: ' + resultsNumber);
}
```

map - compose

Functors – any object or data structure that you can map over

```
var _Container = function (val) {
  this.val = val;
}
```

```
var Container = function(x) { return new _Container(x); }

Container(1);
// > _Container { val:1 }
```

```
_Container.prototype.map = function(f) {
  return Container(f(this.val));
}
```

```
Container("jana").map(capitalize)
// > Container("Jana")

var map = _.curry(function(f,obj) {
  return obj.map(f);
}

map(capitalize, Container("jana");
map(compose(first, reverse), Container("jana"));
```

Maybe

```
_Maybe.prototype.map = function(f) {
  return this.val ? Maybe(f(this.val)) : Maybe(null);
}

map(capitalize, Maybe("jana");
// > Maybe("Jana")

var name = null;
map(capitalize, Maybe(name);
// > Maybe(null)

map(compose(first, reverse), Container("jana"));
```

Either

IO

Future

EventStream

Monads

- Nested computations
  - mjoin
  - chain

```
// nested mappings
var renderPage = compose(map(map(ourRenderFunction), ourPrepareFunction);
var renderPage = compose(mjoin, map(ourRenderFunction), ourPrepareFunction);
```

We can implement mjoin, chain or both

```
var chain = function(f) {
  return compose(mjoin, map(f));
}

var mjoin = chain(id);
```

Example – with monads

```
var matchElement = function(searchWord) {
  return compose(_.curry(matchText)(searchWord), getCommentFromElement);
}

var doSearch = function(searchWord) {
  return _.filter(matchElement(searchWord))(getAllCommentElements());
}

var getResults = compose(mjoin, map(_.forEach(applyResultStyle)), map(doSearch), Maybe, getSearchWord);

function search() {
  var results = getResults("#searchWord")
  var resultsNumberElement = $('#resultsNumber');
  resultsNumberElement.text('Results: ' + results.length);
}
```

Where do we use monads?

- Ajax libraries
- Promise implementations
- JQuery is a monad

We are used to seeing custom names:

- Promise(x).then(f)

While in terms of algebraic functors they are defined generally

- map(f, Promise(x))

Where do we use monads?

- Ajax libraries
- Promise implementations
- JQuery is a monad

We are used to seeing custom names:

- Promise(x).then(f)

While in terms of algebraic functors they are defined generally

- map(f, Promise(x))

References

- ramdajs
  - http://ramdajs.com/repl/?v=0.18.0
  - auto curried functions
  - arguments arranged suitable for currying
- fantasy-land
  - Algebraic JavaScript Specification
  - Lots of modules and libs
  - pointless-fantasy: a point-free implementation
- ttps://github.com/douglascrockford/monad