Introduction to Reason(ML)

Arnoldas Šidlauskas

What is it?

•••

Reason

A new language by Facebook

Based on OCaml

Reason was open sourced a year ago and started ~2 years ago

Lead by Jordan Walke (creator of React)

Syntax heavily inspired by modern JavaScript, Rust

OCaml

Statically typed functional language (ML family)

Pragmatic approach

Created around the same time as JS

Compilation to multiple backends

Used by Jane Street, Facebook

The reason behind Reason

Facebook had a lot of success using OCaml

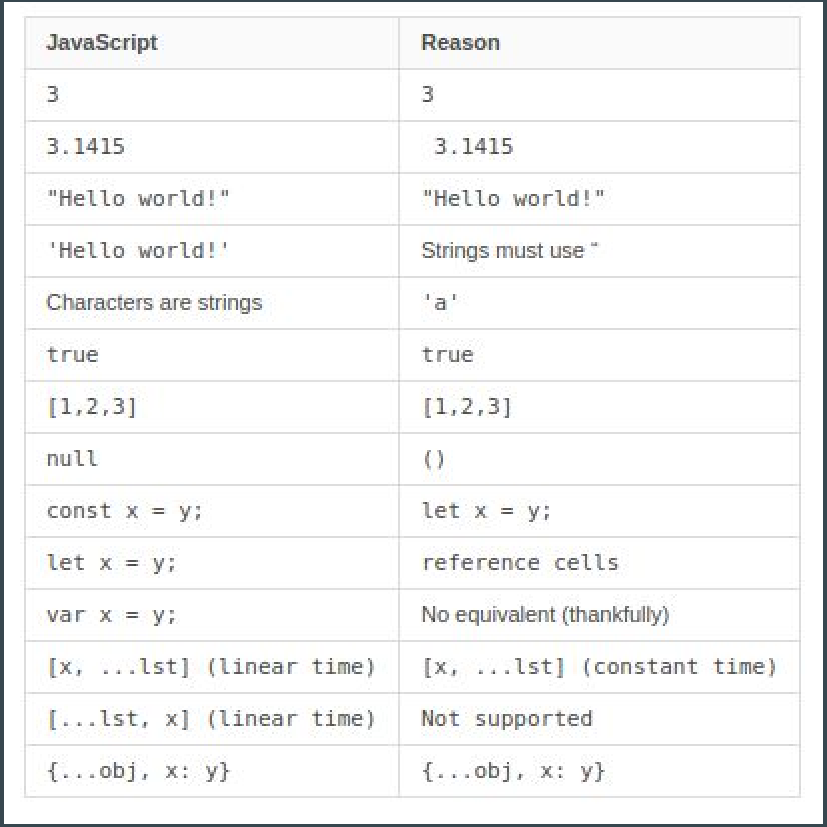
A lot of complaints about syntax

Good fit for web development, but lack of tools/libraries

The language

•••

Syntax



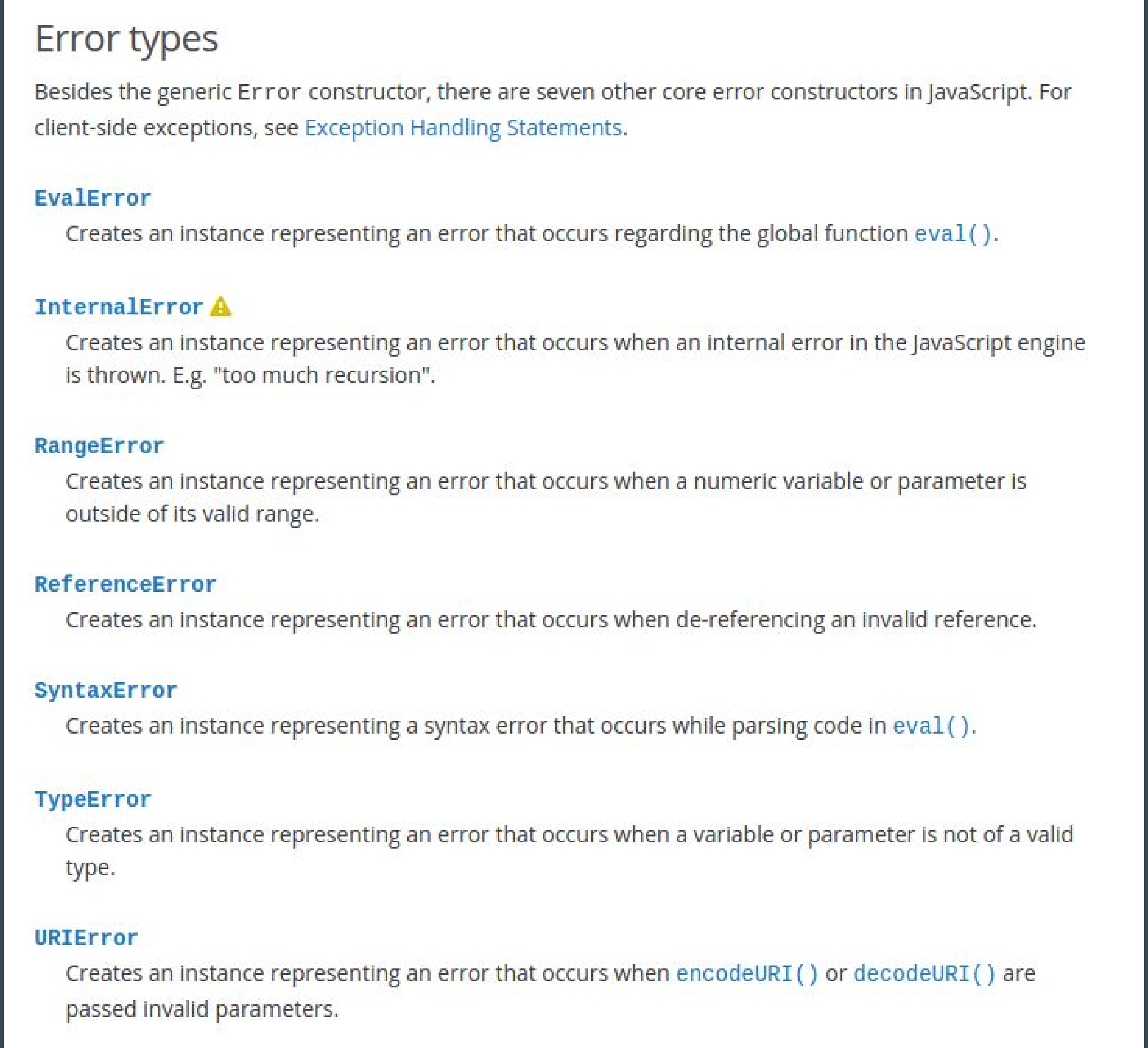
JavaScript	Reason
`3`	`3`
`3.1415`	`3.1415`
`"Hello world!"`	`"Hello world!"`
`'Hello world!'`	Strings must use "
Characters are strings	`'a'`
`true`	`true`
`[1,2,3]`	`[1,2,3]`
`null`	`()`
`const x = y;`	`let x = y;`
`let x = y;`	`reference cells`
`var x = y;`	No equivalent (thankfully)
`[x, ...lst] (linear time)`	`[x, ...lst] (constant time)`
`[...lst, x] (linear time)`	`Not supported`
`{...obj, x: y}`	`{...obj, x: y}`

Static typing

Once it compiles it runs

Helps prevent bugs

Great type inference



Error types

Besides the generic `Error` constructor, there are seven other core error constructors in JavaScript. For client-side exceptions, see Exception Handling Statements.

`EvalError`
Creates an instance representing an error that occurs regarding the global function `eval()`.

`InternalError` ⚠
Creates an instance representing an error that occurs when an internal error in the JavaScript engine is thrown. E.g. "too much recursion".

`RangeError`
Creates an instance representing an error that occurs when a numeric variable or parameter is outside of its valid range.

`ReferenceError`
Creates an instance representing an error that occurs when de-referencing an invalid reference.

`SyntaxError`
Creates an instance representing a syntax error that occurs while parsing code in `eval()`.

`TypeError`
Creates an instance representing an error that occurs when a variable or parameter is not of a valid type.

`URIError`
Creates an instance representing an error that occurs when `encodeURI()` or `decodeURI()` are passed invalid parameters.

First class immutable data structures

Makes code easier to reason about

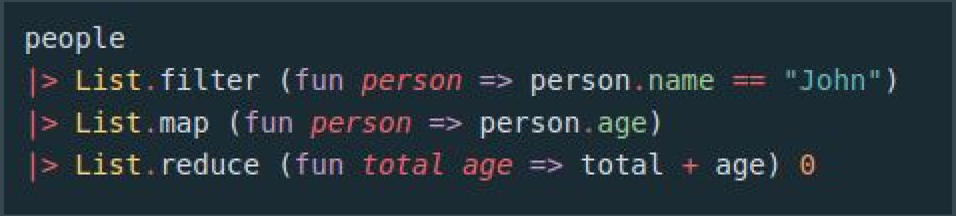
Works well with modern front end frameworks

Makes some things much easier to implement

Will not be coming in JS in the nearby future

Automatic currying of functions

Helps a lot in function composition



```
people
|> List.filter (fun person => person.name == "John")
|> List.map (fun person => person.age)
|> List.reduce (fun total age => total + age) 0
```

Labeled parameters

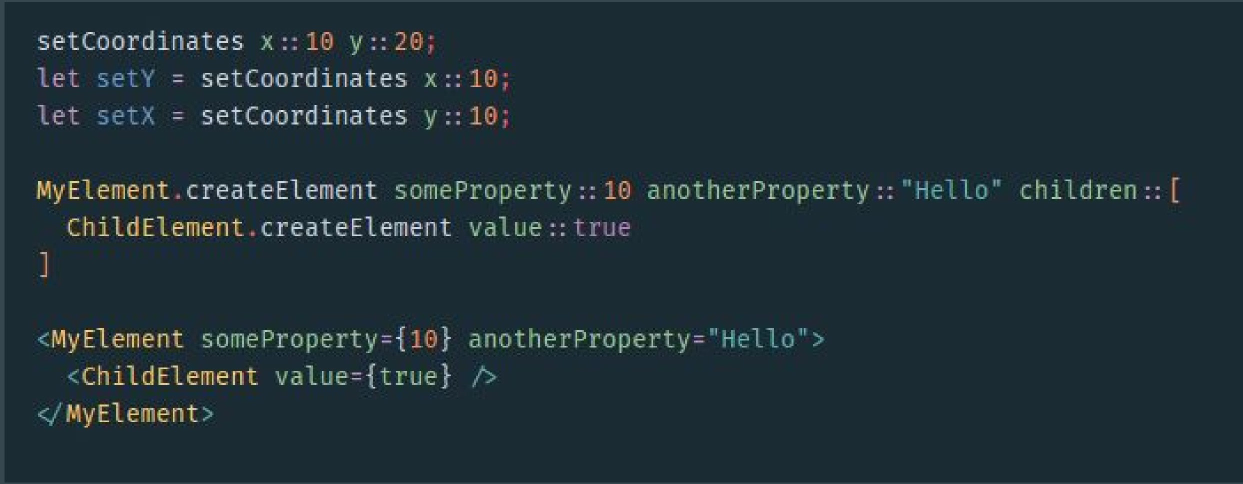
Can make code easier to read/learn (http://worrydream.com/LearnableProgramming/)

Can prevent some bugs

Allows currying in any order

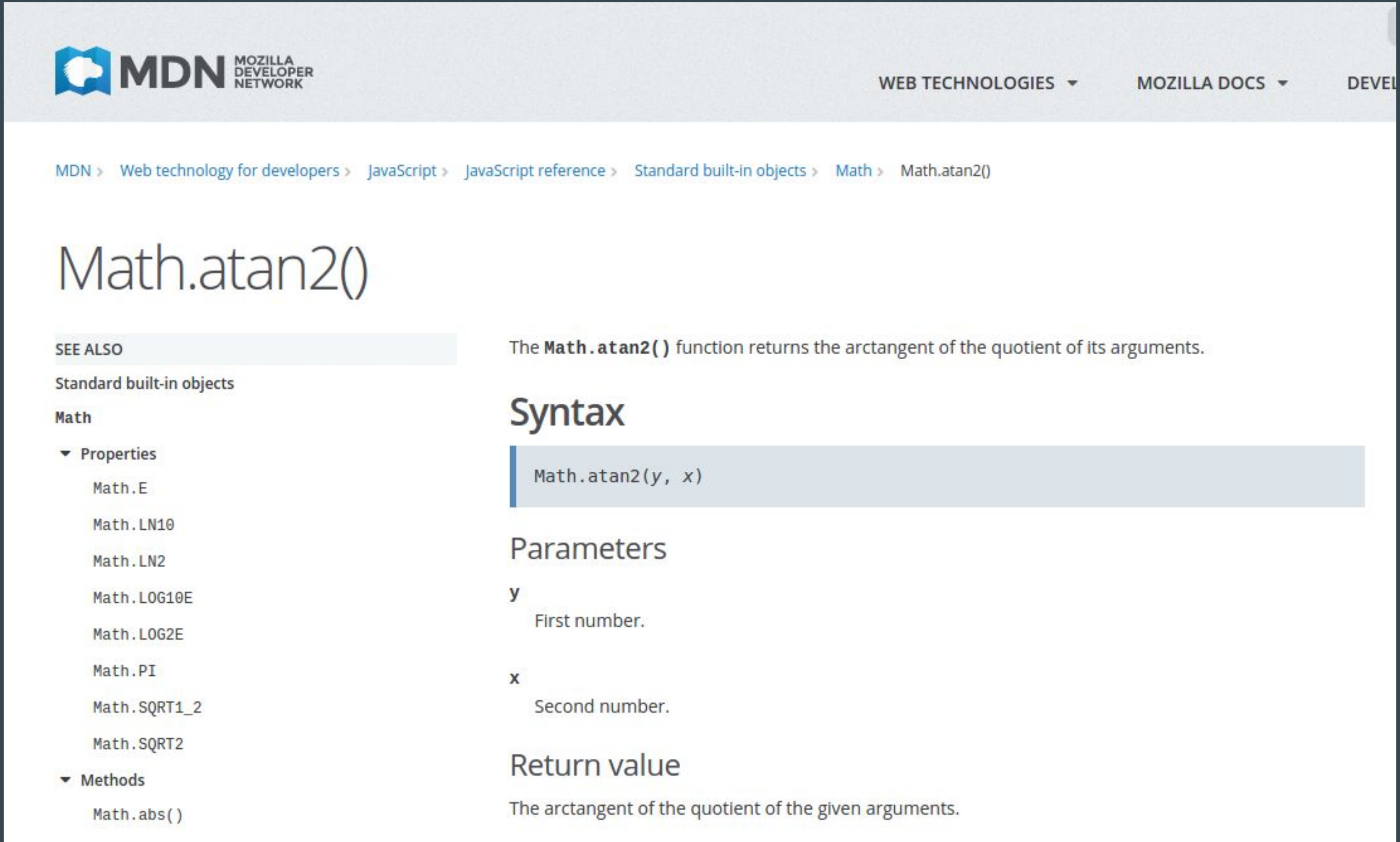
Good fit for UI development

We are used to work around it via various conventions



```
setCoordinates x::10 y::20;
let setY = setCoordinates x::10;
let setX = setCoordinates y::10;

MyElement.createElement someProperty::10 anotherProperty::"Hello" children::[
  ChildElement.createElement value::true
]

<MyElement someProperty={10} anotherProperty="Hello">
  <ChildElement value={true} />
</MyElement>
```

Except when this happens



MDN MOZILLA DEVELOPER NETWORK

WEB TECHNOLOGIES ▾ MOZILLA DOCS ▾ DEVEL

MDN › Web technology for developers › JavaScript › JavaScript reference › Standard built-in objects › Math › Math.atan2()

Math.atan2()

SEE ALSO

Standard built-in objects

`Math`

▾ Properties

- `Math.E`
- `Math.LN10`
- `Math.LN2`
- `Math.LOG10E`
- `Math.LOG2E`
- `Math.PI`
- `Math.SQRT1_2`
- `Math.SQRT2`

▾ Methods

- `Math.abs()`

The **`Math.atan2()`** function returns the arctangent of the quotient of its arguments.

Syntax

```
Math.atan2(y, x)
```

Parameters

y

First number.

x

Second number.

Return value

The arctangent of the quotient of the given arguments.

Module system

Added to JS in ES2015 (aka ES6)

Still not supported by any browser yet

OCaml has a module system that is much more powerful than the JavaScript one

JS Compilation

•••

BuckleScript

Made by Bloomberg

Produces readable code

Really fast compilation time

Aggressively eliminates dead code

Can be used with Google Closure compiler / RollupJS

Reached 1.0 less than a year ago

BuckleScript benchmarks

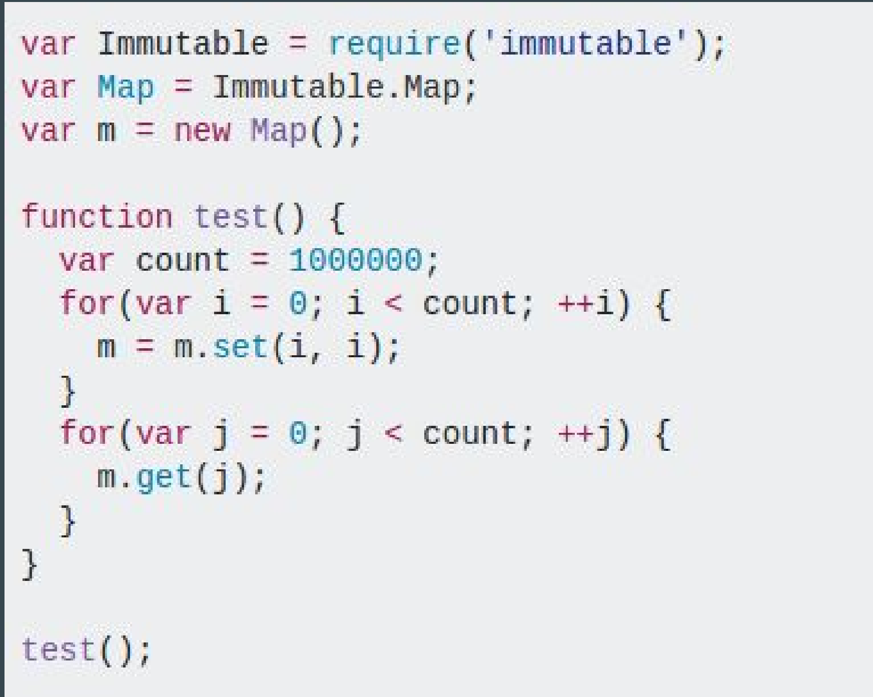
(https://github.com/bloomberg/bucklescript#immutable-data-structures)

Execution Time:

- BuckleScript: 1186ms
- JavaScript with Immutable.js: 3415ms

Compiled Size:

- BuckleScript (production): 899 Bytes
- JavaScript with Immutable.js: 55.3K Bytes



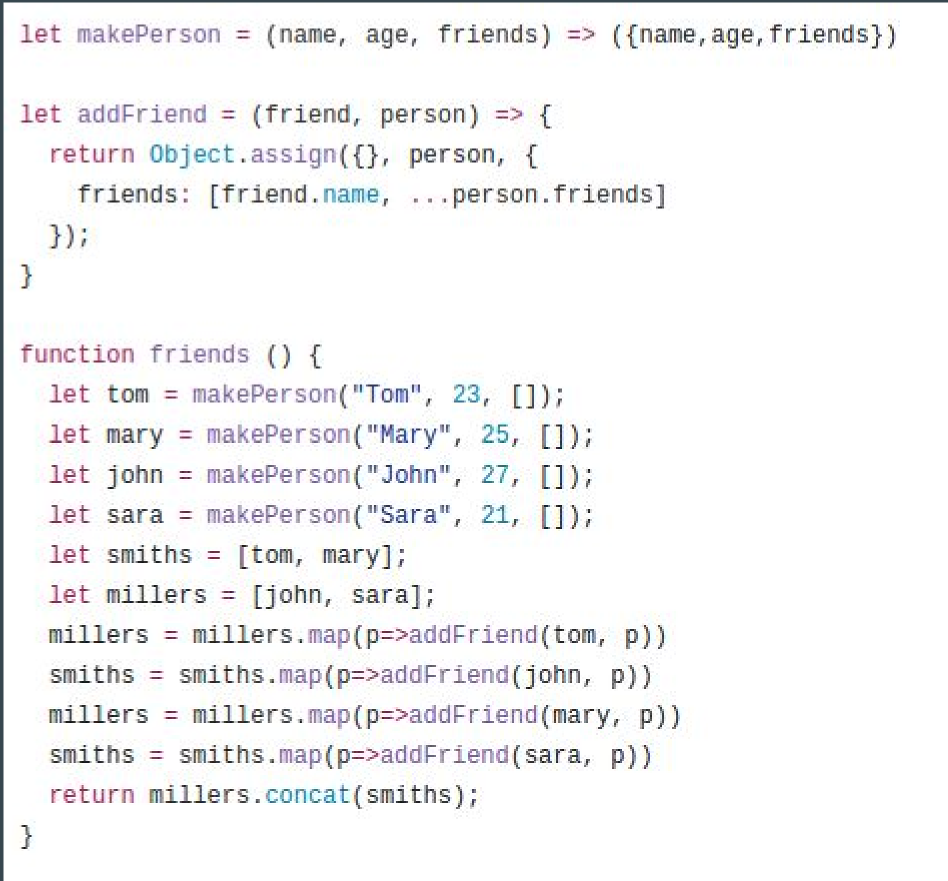
```
var Immutable = require('immutable');
var Map = Immutable.Map;
var m = new Map();

function test() {
  var count = 1000000;
  for(var i = 0; i < count; ++i) {
    m = m.set(i, i);
  }
  for(var j = 0; j < count; ++j) {
    m.get(j);
  }
}

test();
```

BuckleScript benchmarks

Timings:
Reason: using BuckleScript Records/Lists : 710.390ms
JS: using Object.assign   8263.039ms

Timings of unfair/cheating variants
JS: using Object and manual key mapping (brittle code!) : 3123.591ms
JS: using Object mutation (no immutability!) : 1721.166ms

https://github.com/neonsquare/bucklescript-benchmark



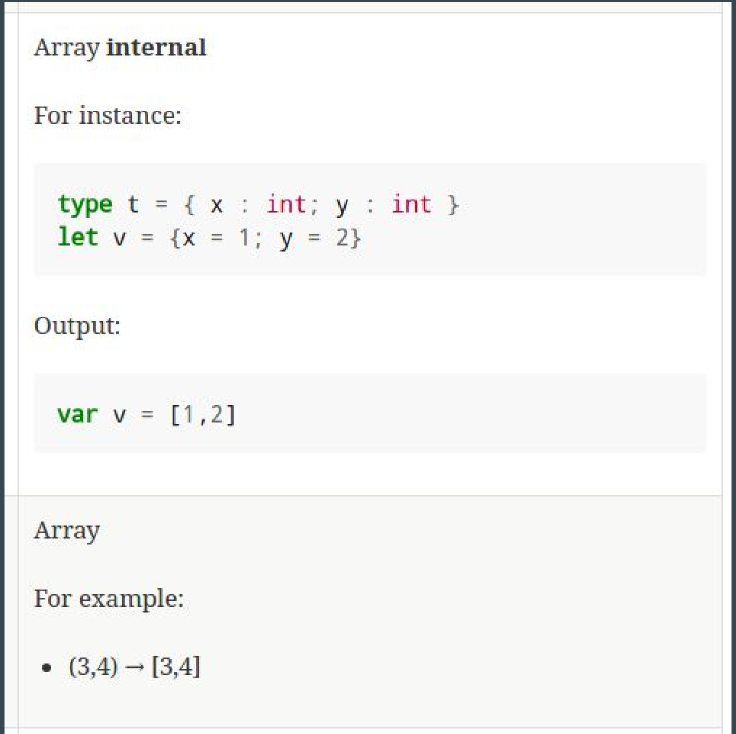
```
let makePerson = (name, age, friends) => ({name,age,friends})

let addFriend = (friend, person) => {
  return Object.assign({}, person, {
    friends: [friend.name, ...person.friends]
  });
}

function friends () {
  let tom = makePerson("Tom", 23, []);
  let mary = makePerson("Mary", 25, []);
  let john = makePerson("John", 27, []);
  let sara = makePerson("Sara", 21, []);
  let smiths = [tom, mary];
  let millers = [john, sara];
  millers = millers.map(p=>addFriend(tom, p))
  smiths = smiths.map(p=>addFriend(john, p))
  millers = millers.map(p=>addFriend(mary, p))
  smiths = smiths.map(p=>addFriend(sara, p))
  return millers.concat(smiths);
}
```

The reason behind this

http://bloomberg.github.io/bucklescript/Manual.html#_runtime_representation

Record

Tuple



Array **internal**

For instance:

```
type t = { x : int; y : int }
let v = {x = 1; y = 2}
```

Output:

```
var v = [1,2]
```

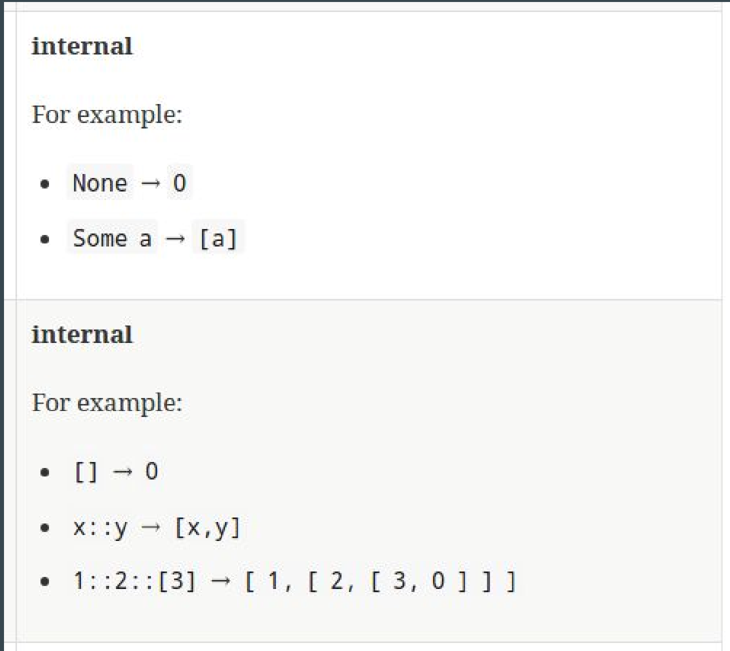
Array

For example:

- (3,4) → [3,4]

Option

List



internal

For example:

- `None → 0`
- `Some a → [a]`

internal

For example:

- `[] → 0`
- `x::y → [x,y]`
- `1::2::[3] → [ 1, [ 2, [ 3, 0 ] ] ]`

```
type person = {
  name: string,
  surname: string,
  hobby: string
};

let p1 = {
  name: "John",
  surname: "Smith",
  hobby: "Baking cookies"
};

let p2 = {
  ...p1,
  surname: "Doe",
};

let p3 = {
  ...p2,
  name: "Ted",
  hobby: "Baking pancakes"
};
```

```
// Generated by BUCKLESCRIPT VERSION 1.7.1, PLEASE EDIT WITH CARE
'use strict';

var p2 = /* record */ [
  /* name */ 'John',
  /* surname */ 'Doe',
  /* hobby */ 'Baking cookies',
];

var p3 = /* record */ [
  /* name */ 'Ted',
  /* surname */ 'Doe',
  /* hobby */ 'Baking pancakes',
];

var p1 = /* record */ [
  /* name */ 'John',
  /* surname */ 'Smith',
  /* hobby */ 'Baking cookies',
];

exports.p1 = p1;
exports.p2 = p2;
exports.p3 = p3;
/* No side effect */
```

JS -> Reason conversion

```
/* original JS file you've copied
over */
const school = require('school');

const defaultId = 10;

function queryResult(usePayload,
payload) {
  if (usePayload) {
    return payload.student
  }
  return
school.getStudentById(defaultId);
}
```

```
/* in a dedicated School.re file */
type student;
external getStudentById: int => student = "getStudentById"
[@@bs.module "School"];
external getAllStudents: unit => array student = "getAllStudents"
[@@bs.module "School"];

/* in the current file */
type payloadType = Js.t {. student: School.student}; /* TODO: put this
somewhere else! */

let defaultId = 10;

let queryResult usePayload (payload: payloadType) => {
  if (Js.to_bool usePayload) {
    payload##student
  } else {
    School.getStudentById defaultId;
  }
};
```

JS -> Reason react conversion with interop

```
module Greeter = {
  include ReactRe.Component.JsProps;
  let name = "Greeter";
  type props = {
    greeting: string,
    children: list ReactRe.reactElement,
  };
  type jsProps = Js.t {.
    greeting: string,
    children: Js.null_undefined ReactRe.reactJsChildren
  };
  let jsPropsToReasonProps = Some (
    fun jsProps => {
      greeting: jsProps##greeting,
      children: ReactRe.jsChildrenToReason jsProps##children,
    }
  );
  let render { props } =>
    <div>
      <div> (ReactRe.stringToElement props.greeting) </div>
      (ReactRe.listToElement props.children)
    </div>;
};
include ReactRe.CreateComponent Greeter;
let createElement ::greeting ::children => wrapProps {greeting, children} ::children;
```

```
class Greeter extends React.Componen {
  static propTypes = {
    greeting: React.PropTypes.string.isRequired,
    children: React.Children,
  };

  render() {
    return (
      <div>
        <div>{this.props.greeting}</div>
        {this.props.children}
      </div>
    );
  }
}
```

There is much more

Good error messages

Sandboxed environments

Refmt

MirageOS Unikernels

Conclusion

Type checking really helps

A lot of awesome features

BuckleScript is fast

Good interoperability with existing JS

Easy to get started, however lots of features = lots to learn

Still a bleeding edge technology, quite a few rough edges

Thank you for your attention

•••

https://facebook.github.io/reason/