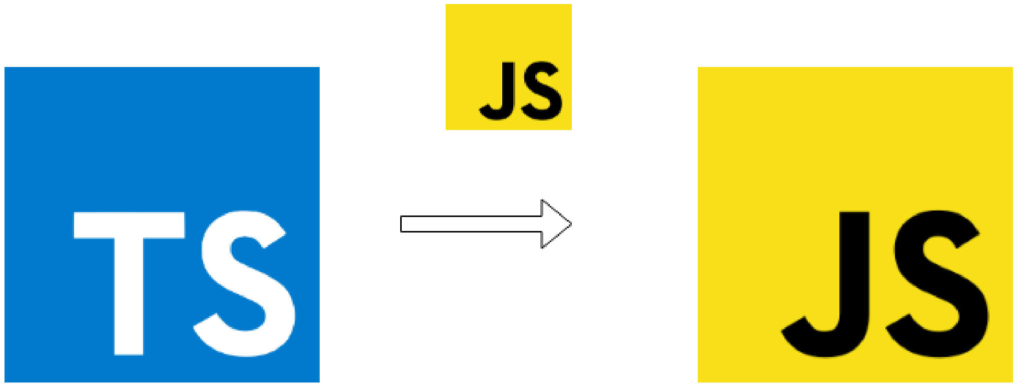
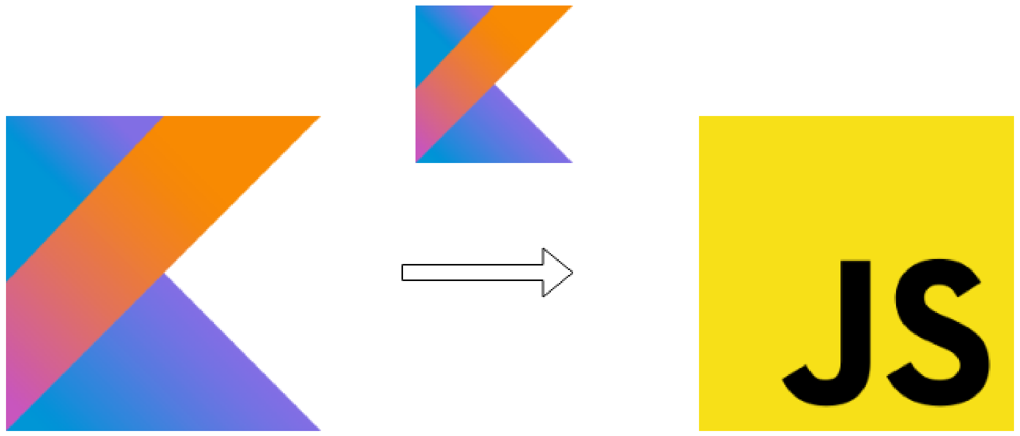
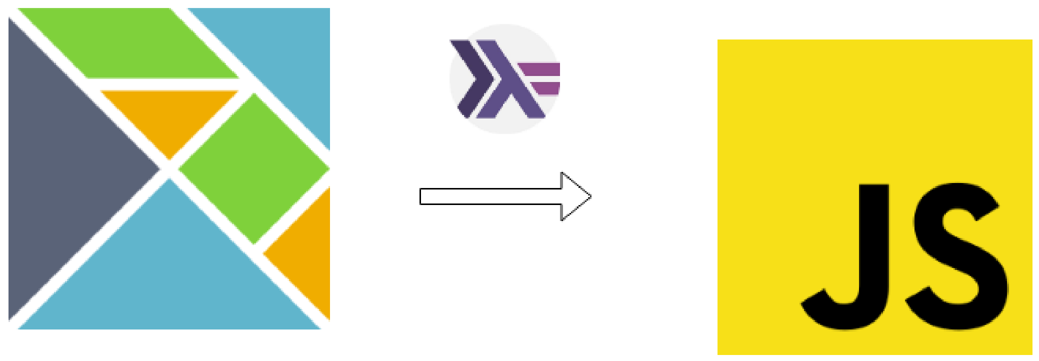
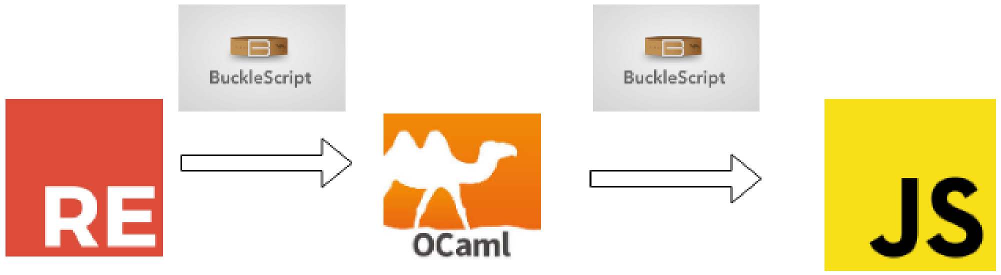
REASONML JAVASCRIPT OCAML

Good Bad and Ugly


TS
JS
JS


JS


JS


RE
BuckleScript
OCaml
BuckleScript
JS

Reasonml

* Static typed
* Functional and OOP
* Not pure
* Immutable
* JSX

Chapter I
Overview

Functions

```
let fn = (a, b) => a + b;

let fn1 = () => {
  let var = 42;
  43;
}

fn1() // 43
```

Key Word Arguments

```
let fn = (~argument1, ~argument2) => argument1 +
argument1;

fn(~argument2=42, ~argument1=7);
```

Data structures

```
let arr = [|1,2,3,4|];
let lst = [1,2,3,4];
let record = {
  attr: "",
  attr1: 42,
  attr2: 4.4,
}
[] == [] // true
```

Pipes

```
let result = [1, 2, 3]
   |> map(a => a + 1)
   |> filter(a => a mod 2 === 0);

let result1 = , filter(a => a mod 2 === 0, ([1, 2,
3], map(a => a + 1)));
```

Curring

```
let add = (a, b) => a + b;
let add1 = add(1);
let result = add1(2);
```

Chapter II
Type system

Usage based types

```
let add = (a, b) => a + b;

add(42, []); // This has type: 'a list But
somewhere wanted: int
```

Usage based types

```
let addFloats = (a, b) => a +. b;
let addStrings = (a, b) => a ++ b;
```

Usage based types

```
let add = (a: int, b: int): int => a + b;
```

Records

```
let get = () => ({
  attr: "",
}) // The record field attr can't be found.
```

Records

```
type r = { attr: string };

let get = () => ({
  attr: "",
});
```

Variants

```
type animal = Dog | Cat | Bird;
```

Chapter III
Pattern matching

Absence handling

```
type option('a) = None | Some('a);

let fn = (maybeItem: option(int)) => switch
(maybeItem) {
  | None => 0
  | Some(item) => item
}
```

Value based

```
let fn = (a: int) => switch (a) {
  | 0 => 0
  | 1 => 42
  | _ => 99999
}
```

Value based

```
let fn = (items: list(int)) => switch (items) {
  | [] => 0
  | _ => 99999
}
```

Recursive

```
let rec sum = (items: list(int)) => switch (items)
{
   | [] => 0
   | [head, ...tail] => head + sum(tail)
}
```

Exception handling

```
switch([1, 2, 3, 4] |> List.filter(x => x mod 5
=== 0)) {
   | item => "Found"
   | exception Not_found => "Not found"
}
```

Differences

* No try catch (+/-)

* No optional

* No Union

* no interfaces (weak polymorphism)

Chapter IV
Function however...

References

```
let foo = ref(5);
let five = foo^; /* 5 */
foo := 6;
```

OOP

```
type tesla = {.
  drive: int => int
};

let obj: tesla = {
  val hasEnvy = ref(false);
  pub drive = (speed) => {
    this#enableEnvy(true);
    speed
  };
  pri enableEnvy = (envy) => hasEnvy := envy
};
```

Mutable

```
type node = {
  name: nodeName,
  text: string,
  mutable position: int,
  attributes,
  handlers: list((string, option(eventHandler))),
  children: list(node),
};
```

Imperative

```
let start = 1;
let end = 3;

for (x in start to end) {
  print_int(x);
};
```

Differences

* Focused on on best practices
* Allow to write other paradigms
* But is it more difficult

Chapter V
Usage

Reason React

```
module App = {
  let component =
ReasonReact.statelessComponent("App");
  let make = _children => {
    ...component,
    render: self =>
      <button> (ReasonReact.string("Hello!"))
</button>,
  };
};

ReactDOMRe.renderToElementWithId(<App />, "id")
```

REVERY

Revery UI

```
let init = app => {
  let win = App.createWindow(app, "test");

  let textHeaderStyle =
    Style.[
      backgroundColor(Colors.black),
      color(Colors.white),
      fontFamily("Roboto-Regular.ttf"),
      fontSize(24),
    ];

  let render = () => {
    <view
      style={Style.[
        position(Absolute),
        bottom(10),
        top(10),
        left(10),
        right(10),
        backgroundColor(Colors.blue),
      ]}>
      <view

        style={Style.[
```

Rembrandt

```
open Rembrandt.Elements;

type model = int;
type action =
  | Add
  | Sub
  | Twice;

let update =
    (model: model, action: action): (model, Command.command('action))
=>
  switch (action) {
  | Add => (model + 1, Command.null)
  | Sub => (model - 1, Command.null)
  | Twice => (model + 1, Command.action(Add))
  };

Rembrandt.run( ~model=42, ~update, ~view= (model, dispatch) =>
  <div>
    <div id="count"> {string_of_int(model) |> text} </div>
    <button id="plus" onClick={_ => Add |> dispatch}>
      {text("+")}
    </button>

    <button id="minus" onClick={_ => Sub |> dispatch}>
```

REASON PROS

* Great type system
* Functional and immutable by default
* Facebook support
* Optional imperative paradigm
* JS syntax
* JS/TS ports

REASON CONS

* Forced variant way

* Hard to debug

* Much less popular than TS

* No reactive libs

* Ticky syntax

LINKS

* https://egghead.io/courses/get-started-with-reason
* http://2ality.com/2017/11/about-reasonml.html
* https://reasonml.github.io/
* https://reasonml.github.io/reason-react/
* https://github.com/revery-ui/revery
https://github.com/przemyslawjanpietrzak/rembrandt/

Thank you :*