What is ReScript?

Written by Seth Corker on Benevolent Bytes

ReScript is a rebranding of BuckleScript that is trying to embrace the JS side of things rather than OCaml. Reason, formally ReasonML, and BuckleScript has been around for a few years. Reason is a programming language that leveraged OCaml and JavaScript to offer an easy to learn syntax for JavaScript developers and offer some strong type guarantees. BuckleScript is used to produce JavaScript and offer a set of useful libraries. The idea of ReScript is to consolidate these tools into something a little more coherent so as a developer you're not trying to figure out how OCaml, BuckleScript and Reason fit together.

Why the rebrand?

The ReScript team summed it up on their blog post. The idea is to aim for a more unified experience. Reason still exists but ReScript aims to embrace the JS ecosystem and optimise for that use case versus the in-between JavaScript and OCaml. What this means in practice is an easier to understand brand. ReScript is a language that produces JavaScript and is strongly typed. That's the selling point and that's what the rebrand is trying to convey. It's for developers looking for something beyond JavaScript in a more functional style than TypeScript. ReScript is shedding the cognitive load of the tools on which it relies.

What does ReScript look like?

Right now, it looks a lot like Reason but as time progresses, that's likely to change. It's embracing the a syntax that is more comfortable than Reason for JavaScript devs.

Let's take a look at three samples of code.

ReScript

```
module Button = {
  @react.component
  let make = (~count: int) => {
    let times = switch count {
    | 1 => "once"
    | 2 => "twice"
    | n => Belt.Int.toString(n) ++ " times"
    }
    let msg = "Click me " ++ times

    <button> {msg->React.string} </button>
  }
}
```

As a JavaScript developer who uses React, this isn't too challenging to understand what's going on. We're making a button component that shows how many times the button has been clicked which comes through as a prop called `count`.

Reason

```
module Button = {
  [@react.component]
  let make = (~count: int) =>
    [@ns.braces]
    {
      let times =
        switch (count) {
        | 1 => "once"
        | 2 => "twice"
        | n => Belt.Int.toString(n) ++ " times"
        };
      let msg = "Click me " ++ times;

      <button> msg->React.string </button>;
```

```
    };
};
```

The Reason syntax is quite similar to ReScript but there are a few more square brackets and semicolons. Not too bad.

JavaScript (Generated)

```
import * as React from "react";

function Playground$Button(Props) {
  var count = Props.count;
  var times = count !== 1 ? (
      count !== 2 ? String(count) + " times" : "twice"
    ) : "once";
  var msg = "Click me " + times;
  return React.createElement("button", undefined, msg);
}

var Button = {
  make: Playground$Button
};

export {
  Button ,
}
```

This is the generated JavaScript. It's readable, if you were writing it by hand you'd probably opt for JSX but it's not obfuscated and you can follow what's going on.

JSX

```
import * as React from "react";

export function Button({ count }) {
  const times = count === 1 ? "once" : (count === 2 ? "twice" : `${count} times`)
  return <button>Click me {times}</button>;
}
```

If I were to write this by hand in ES6, this is what it might look like.

Why would I use ReScript?

You might be wondering what's the point in learning ReScript at all. I'm right there with you. I've used Flow and TypeScript before and a lot of the same benefits can be found in ReScript. For most developers, there's not a good reason to switch over to ReScript other than to try it out. Maybe there will be some killer features one day but right now there is nothing that sways me. It's also going to be an uphill battle if ReScript wants to compete with TypeScript due to its thriving community. So what does ReScript do that TypeScript doesn't.

What does ReScript do differently?

There are two selling points I see so far, again they're not ground breaking but they are a point of interest for the language.

Build Performance ReScript is fast to compile. If you're used to lengthy Webpack builds that we've come to expect of modern tooling, ReScript will look like lighting. The build system is capable of handling projects of very large sizes quite quickly. The benchmark on the ReScript docs highlights a 10,000 file build which takes around 3 minutes. Generally, the small projects I've done with a handful of have taken less than 100ms and incremental builds are a fraction of that. This is a cool feature of the build system that Reason also relies on but it's becoming less of a killer feature as tools like esbuild and vite become more popular.

TypeScript generation This is probably the best features of ReScript because the likelihood of going from vanilla JS to ReScript is low. Most likely you're already using TypeScript and are looking for some more functional features like pattern matching, better type inference and a more robust type system. The ability to drop in some ReScript files to an existing project is a good way to try and improve adoption and doesn't lock you in.

Should I use it?

Probably not. If you're already using Reason for web projects then ReScript is set to make that experience a much more cohesive one. If you're already using static type checking in your project then it's not really worth the effort. That's not to say ReScript doesn't have potential but I don't think it's ready for prime time yet. I'm happy to see where it goes and what the ReScript team comes up with, maybe given some more time it will be worth another look. At the time of writing, ReScript is currently on v9.1 with v10 planned for early 2021. I'll check back and see what's going on with the language and how it's evolving.