Challenge: Program without variables #javascript

hackernoon.com/challenge-program-without-variables-javascript-bee89a41455e

The challenge is to create a piece of software (something real, tangible and more than a hello world) without any variables.

The idea came from a tweet of Samer Buna's article. Which I responded (in jest) with try to "code without variables".

I am planning on creating a Promise library. I was inspired by an article I stumbled across by Trey Huffine, Learn JavaScript Promises by Building a Promise from Scratch.

I believe Promises are simple enough to understand the code and also complex enough to be a challenge.

Warning!

If I were to compare writing clean code to riding the Tour de France, *this code is not that*. This code would be better compared to the X-Games BMX Freestyle Big Air. You about to see a couple of double backflips and 360s, but when *you* get on *your* bike it's probably best to keep all wheels on the road. Kids, don't try this at home *or work*.

With that being said (*if you allow yourself*) there is a lot to learn from this code and I encourage you to create your own playground and see how extreme and freaky you can get. It's at the edges is where you'll discover the most interesting things.

The Rules

- Keywords not allowed: `var`, `let`, `const`, `import`, `class`. Bonus points for not using `if` or `switch`, or `function` keywords.
- Libraries are allowed as long as all rules are followed.
- New libraries can be created, but must follow all rules.
- Libraries must be generic enough to use in any project and cannot be a substitution for the business logic of the code created.
- Tests are not necessary. But if you choose to write tests, they are not subject the rules.

Let's Start!

TDD

A Promise library is farily complex and as I make changes to the code, I want to be sure those changes don't break anything that previously was working. So I am going to start by writing out all my tests first. This is made easy because node already includes a Promise library, so I will write my tests against that first.

One difference, is I do not plan to create any classes as **I find classes unnecessary in JavaScript**. So instead of the typical code you would use to create a Promise: `new Promise((resolve, reject))`, You can just use `XPromise((resolve, reject)),` excluding the `new` keyword.

XPromise.tests.js

Start with the interface

Right away I was presented with a challenging task. Similar to the A+ Promise implementation, I wanted to be able to create a Promise using `XPromise((resolve, reject) => ...)`, `Promise.resolve(...)` and `Promise.reject(...)`. So `XPromise` needs to be a function, but also have 2 properties (`resolve` and `reject`), which are also functions.

Normally this would not require much thought, but because of the rules I am unable to do something like this:

Time to get creative by using `Object.assign` to attach `resolve` and `reject` to the main function.

So far I am pretty happy with this. That is, until I realize `resolve` and `reject` are helper functions that will eventually need to be pointed to the main `XPromise` function, which now there is no reference to ☹

Creating **a reference without a variable**

`XPromise` also needs to return an object that contains 2 functions, `then` and `catch`. Those functions must call the original `XPromise` function, which (again) there is no longer a reference to.

So... I need figure out how to create an asynchronous, recursive, anonymous function or this whole thing is gonna be a bust. Crap.

It's time to bust out the Combinators

When talking about anonymous recursive functions, the famous Y combinator immediately comes to mind. That is the Y Combinator's purpose. Though, the Y Combinator isn't the only combinator we can use. For this task, I have decided to use the much less known, but more simple U Combinator.

I dig the U Combinator because it's easy to remember.

```
f => f(f)
```

That's it! The U Combinator takes a function as an argument and then passes that function to itself. Now the first argument of your function will be your function. If that sounds confusing, *that's because it is confusing*. Don't worry about that, it'll be easier to see in code.

Take notice of the part `sayHello => () => 'hello'` and how it is the same for both `sayHello` and `UsayHello`.

Now let's try this with recursion.

Perfect! This exactly what we need! Now it's time to jam it into the project.

Okay, this is the basic skeleton of a Promise. We have our main function `XPromise`, the helper functions `resolve` and `reject`. `XPromise` takes a function, which contains `resolve` and `reject`. This function also returns an object that contains the functions `then` and `catch`.

You can see that I am also using an Immediately-Invoked Function Expression to make the U Combinator available to use as the `U` argument.

Stay with me now, the worst is over! If I haven't lost you and are still following... rest assured, for the remainder of this article we'll be coasting down hill! 😀

Storing state

This application, like others has to store some kind of state. This will either be the values from `resolve`, `reject` and/or the functions from `then` and `catch`. Typically, this would be done with good 'old variables. Though, we can also accomplish the same thing just using default parameters. This will also give the added benefit of being able to call the function and also seed it with a new state! Which, spoiler alert, we are going to do just that!

BTW, This is a great case for Redux!

Converting blocks to expressions

I *prefer* coding with **expressions**over **blocks**. This is just a preference of mine. `if` statements and `try/catch` contain blocks, so I gotta make an alternative. I also like to use the **comma operator** to combine multiple expressions, which you will see below.

Ya, that is much better! 😀

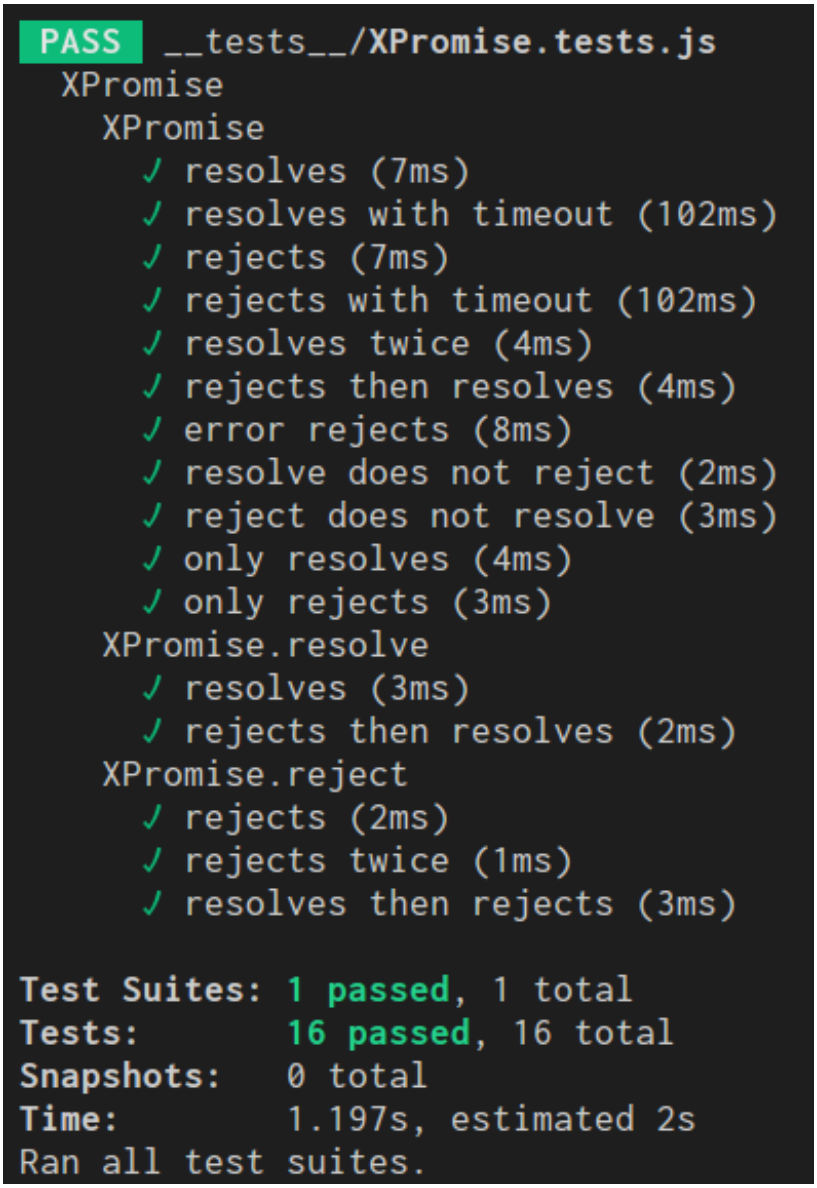
Now I wanna clean up that `try/catch`.

Fast forward >>

This article is about the challenge of writing software without using variables. **This article is not about how to create a Promise library.** So to save time, let's skip the boring shit and just fill in the blanks.

Well, there it is, in all it's glory; A Promise library without a single `const`, `let`, or `var`.

And check this out... all my tests are passing! 😁



Source Code @ https://github.com/joelnet/XPromise

Postgame wrap-up

This challenge ended up being a lot harder (time consuming) than I thought. Not necessarily because of the limitations, but because **creating a Promise library was much more complex than I expected it to be**. A promise may or may not be called synchronously/asynchronous, may or may not have a resolve value, a reject value a then resolver and/or a catch resolver. That is 64 possible states! Clearly I don't have enough tests.

I finally had a legitimate use case for the U Combinator, which was totally awesome.

Medium sucks at code blogging and it’s a pain in my ass to create a gist for every tiny snippet ☹

I kind of like the way this library turned out. The entire thing ended up becoming a single expression.

My message to you

I know this became complex fast, don’t feel like you have to understand 100% of it. I just hope that you found it entertaining. I hope you there was something in this article you haven’t seen before. I hope I made you curious to code-explore on your own!