Lesson 6.1 - New string functionality

With ES2015 comes a powerful new feature to handle interpolation with strings. Generally, it is referred to as `template literals`. Let's get started shall we?!

Multi-line strings

One of the first neat things about template literals is that they can span multiple lines:

```
let myString = `
  This is a multiline string,
and it will preserve whitespace
    and newline characters
`
```

As you can see, this makes for creating long strings with newlines and whitespace a very easy thing to do. So easy in fact you can accidentally add whitespace when you didn't mean to, so be careful!

String interpolation

Another super cool feature that has been a long time coming is interpolation. Given an object `person`:

```
const person = { name: 'Fred Flinstone', email: 'fred@slaterock.com' }
```

We can use it in a string like so:

```
const welcome = `Welcome ${person.name}! We will be sending you a welcome email to ${person.email}
```

Awesome! Much nicer to read than a series of concatenated strings, don't you think?

It's also worth noting that anything inside the `${ }` syntax will simply be evaluated as JavaScript so you can 'literally' put anything you want in there and it will be evaluated:

```
> console.log(`${2 * 2}`)
-> 4
```

Custom interpolation

You can also use custom interpolation to create interpolated 'functions' with custom behavior:

```
strip`
   Welcome to Slate Rock ${person.name}
   We have sent your welcome package to ${person.email}
`
->
Welcome to Slate Rock Fred Flinstone
We have sent your welcome package to fred@slaterock.com
```

Wait a minute? What the heck is going on here? Ok let's break it down. The first thing we need is a function called `strip` that takes multiple arguments. Let's create this function. The purpose here will be to strip all surrounding white space from the multiline string.

```
export const strip = (pieces, ...values) =>
  pieces
    .reduce((concatStr, piece, indx) => `${concatStr}${piece}${values[indx] || ''}`, '')
    .replace(/^\s*/gm, '');
```

OK, we are actually calling this `strip` function with the template literal as the argument. Behind the scenes the template literal is being dissected into several parts. The first is an array of strings that is split anywhere the interpolation syntax is found. The `rest` of the arguments are the pieces of JavaScript themselves that have been evaluated.

Let's take this string for example:

```
`Your name is ${person.name}`
```

That will produce the following arguments:

```
strip(['Your name is', ''], 'Fred Flinstone')
```

So what our little `strip` function is doing is grabbing all of the pure strings in the first argument, and then collecting the rest of the arguments into an array called `values`. We can then build the string ourselves (really, this is what JavaScript is doing behind the scenes) and add any extra functionality that we need to!