AML: Alien Markup Language

Introduction

In the year 2018, radiation from a solar flare mutates the Hubble Space Telescope allowing us to receive data being transmitted on the extra-terrestrial Internet. After observing transmissions for some time, we have established that our cosmic neighbors use a markup language that is notably different from HTML. In particular, it is not strictly hierarchical, an element may extend past the closing of the element in which it was opened. We want to build a javascript component that will translate AML into HTML.

About AML

Like HTML, AML has matched pairs of opening and closing elements, indicating that the text between opening and closing elements should have a particular text effect applied. All AML markup elements begin with a caret (^), optionally are followed by an exclamation point indicating it is a closing element, and are followed by a single character effect type indicator. For a string to be valid AML, all elements that are opened must be closed before the end of the document, and bare caret (^) characters (not part of a markup element) are not permitted.

Unlike HTML, AML elements that are opened within a different AML effect do not need to be enclosed by that effect.

Ok: `^~ Greetings ^% Earthling. ^!% ^!~`

Also Ok: `^~ Greetings ^% Earthling. ^!~ How are you? ^!%`

However, a single AML tag cannot be opened twice without being closed.

Not Ok: `^~ Greetings ^~ Earthling. ^!~ ^!~`

The two AML elements identified so far are summarized in the following table:

Text Effect	**Element Opening**	**Element Close**
Strong	^%	^!%
Emphasis	^~	^!~

For example, the AML:

```
Greetings ^%from ^~Glornix^!% Beta-Nine^!~.
```

Would be rendered as:

Greetings **from *Glornix*** *Beta-Nine*.

The Task

Your task is to write a javascript file that defines a global object with a callable method named `translate`.

`translate` should take a string of valid AML as its sole argument, and return a valid HTML fragment with no wrapping elements. It is not necessary to validate the provided AML, but valid AML should result valid HTML fragments. So translating the above example into HTML with the desired text effects applied looks like the following:

```
var myHTML = AMLTranslator.translate("Greetings ^%from ^~Glornix^!% Beta-Nine^!~.")
```

and myHTML would contain the string:

```
Greetings <strong>from <em>Glornix</em></strong><em> Beta-Nine</em>.
```

Other examples:

AML String	**HTML String**
Hello, Earth!	Hello, Earth!
Hello, ^%Earth^!%	Hello, Earth!
^~Hello, ^%Earth!^!~ We are pleased ^~to^!% meet you.^!~	Hello, Earth!We are pleased to meet you.

We have provided you with a sample Node.js program that can be used for evaluating your translator (`aml_tester.js`), read through it, we will test your result by running:

```
> npm install mock-browser
```

```
> node ./aml_tester.js ./YOUR_MODULE.js
```

To get started, create a javascript file that looks like the following:

```
var AMLTranslator = (function() {
  // YOUR CODE GOES HERE.
}());

// Make translator available via "require" in Node.js
if (module.exports) {
  module.exports = AMLTranslator
}
```

When you are finished, send us your completed code.

How We Evaluate Solutions

When we evaluate your solution, we will be asking ourselves the question, "Is this the kind of code we want in the RigUp code base?" Specifically, this means we will be evaluating:

- **Correctness**
  Does this code produce correct results? Correctness is one aspect of customer-facing quality, and we believe that code that consistently produces the expected results without throwing errors or exceptions is an important part of delivering a product that will delight the customer.
- **Testability**
  How easy would it be to write unit tests for this code? We believe testing is integral to both rapid development and to delivering a high quality product to the customer.
- **Readability**
  Is this code easy to understand? Code that is clear is easier to change and easier to test, and it just more pleasant to work on.
- **Extensibility**
  How hard will it be to add features to this code? There are some types of future features that are easy to anticipate. In the case of AML, one could anticipate in the future needing to add additional elements that follow the same rules as elements we have already seen, for example adding an element that creates the text effect of underline. But we have no idea in what way future elements might violate the assumptions presented here, so extensibility for the unknown future also means writing something simple, correct, testable, readable.

Hints

- We're not trying to trick you on the AML parsing. If you've thought up some difficult-to-parse edge case, make your best judgement about how to handle it and put a comment about it in your code.