AML: Alien Markup Language

In the year 2020, radiation from a solar flare mutates the Hubble Space Telescope and allows us to receive data being transmitted on the extraterrestrial Internet. After observing transmissions for some time, we have established that our cosmic neighbors use a markup language that is similar to HTML, but that has some notable differences. In particular, it is not strictly hierarchical: an element may extend past the closing of the element in which it was opened. We now need 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 the opening and closing elements should have a particular text effect applied. All AML markup elements begin with a caret (^), which is optionally followed by an exclamation point (!) to indicate it is a closing tag, and end with a single character indicating the text effect. 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 that are not part of a markup element are not permitted.

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

> **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. ^!~ ^!~`

Two AML elements have been identified thus far:

Text Effect	Opening Tag	Closing Tag
Strong	^%	^!%
Emphasis	^~	^!~

So, for example, the AML string:

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

would be rendered as:

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

Your Task

We would like you to write a javascript module that defines a global object with a callable method named `translate`. The method `translate` should take a string of valid AML as its sole argument, and return a string containing valid HTML with no wrapping elements. It is not necessary to validate the provided AML, but valid AML should always result in valid HTML.

For this task, we are looking for a pure javascript solution, preferably contained in a single file. In particular, *you should not use any external modules, libraries, executables, etc.* You don't need any to solve this problem.

Here are some examples of valid AML inputs with their valid HTML outputs:

Input: `"Hello, Earth!"`
Output: `"Hello, Earth!"`

Input: `"Hello, ^%Earth!^!%"`
Output: `"Hello, <strong>Earth!</strong>"`

Input: `"^~Hello, ^%Earth!^!~ You are ^~welcome^!% here.^!~"`
Output:
```
"<em>Hello, <strong>Earth!</strong></em><strong> You are
<em>welcome</em></strong><em> here.</em>"
```

We have provided you with a test harness (`aml_tester.js`) that can be used to evaluate your translator using Node.js. We will be using this test harness to test your code, so make sure your code passes it. Keep in mind that as part of our evaluation of your solution's correctness and extensibility, we will running it against additional test strings as well. In developing your solution, you may want to add some of your own tests to the harness.

You can run your translator with the test harness this way:

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

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

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

if (module.exports) {
      module.exports = AMLTranslator;
}
```

Note that we're not trying to trick you in any way with this challenge. If you've found some difficult to parse edge case, make your best judgement about how to handle it and put a comment about it in the code. Also feel free to contact us with any questions, we're happy to clarify the problem statement.

When you are finished, upload your completed code using the link in the email. We ask that you not share this code challenge or your solution with other people, and particularly request that you don't post either to the Internet.

How We Evaluate Solutions

When we review your code, the main question we will ask ourselves is "*is this the kind of code we want in the RigUp code base?*" Keep in the mind that we are considering all aspects of your solution and how you coded it, and not just whether it passes the test harness. Specifically, we will be evaluating:

Correctness
Does the code produce correct results? Correctness is obviously a key aspect of customer-facing quality, and we need engineers who write code that consistently produces the expected results without error or exception.

Testability
How easy would it be to write unit tests for this code? Automated testing is integral to both rapid development and to delivering a high quality product to the customer.

Readability
Is this code easy to understand, both now and six months later? Code is that easily readable is easier to change and easier to test, and is just more pleasant to work on.

Extensibility
How hard will it be to add features to this code? These are some types of potential features that are easy to anticipate. In the case of AML, one could anticipate needing to add in the future additional elements that follow the same rules as elements we've already seen, such as one that creates an underline text effect.