JS Template Literals

If you've worked with Javascript for a while you've probably hit the nightmare of string concatenation and how error prone the process is and how hard it is to troubleshoot if you're not careful.

```
var sentence = 'This is a very long sentence that we want to '
 + 'concatenate together.'
```

In ES6/ES2015 there is a new way to create interpolated strings: Template String Literals.

> In this post we'll discuss three areas:
>
> - How to build template string literals and multi line string literals
> - Variable interpolation
> - Possible ways to use template string literals for localization

None of these things are new. You've always been able to do in Javascript with template literals it's now easier and more efficient to do so.

Building Template Literals

To build a Template Literal use the backtick (`) character to open and close the string. In essence it's not different than concatenating strings but it allows you to create the string literals without worrying about whether you interpolated the correct type of quotation mark (' and ") or the + sign when creating multi-line strings.

> At it's simplest, an ES6 Template String Literal is written like this:

```
let demoString = `Hello World.`
console.log(demoString);
```

We can also create Template Literal Strings using the same system. Also note how we've been able to add amgle brackets to open and close tags and single quotation marks to the longer example.

```
let longerString = `The second, greyed out example shown here shows the fo
console.log(longerString);
```

Variable interpolation

Where the Template String Literals really show their strengths is when we interpolate variables in the longer string. Going back to our first example we'll add a variable to show the name of the person we're greeting:

```
var userName2 = "carlos"
var greeting = `Hello, ${userName2}!`
console.log(greeting);
// Returns Hello, carlos!
```

In this example, the interpolation is the `${userName}` string that will take the value of the corresponding variable and put it's name in place holder.

We can also work with arrays as the source of interpolation data, something like the example below where we interpolate the valies in the `userData` array:

```
var userData = {
  "name": "Carlos",
  "home": "Chil"Carlos"var greeting2 = `Hello ${userData.name}!.

The weather in ${userData.home} is...`;
console.log(greeting2);
```

Using that last bit of code we can visit an interesting idea. Using String Template Literals when doing Translation.

```
var userData = {
  "en": {
    "chapter": "Chapter",
 "chapter"on": "Section",
  },
```

```
  "es": {
    "chapter": "Capítulo",
    "se": "n": "Sección",
  },
};

var chapterHeading = ": "`${userData.en.chapter} 1, ${userData.en.section}
console.log(chapterHeading);
// Produces: Chapter 1, Section 1.

var chapterHeadingEs = `${userData.es.chapter} 1, ${userData.es.section} 1
console.log(chapterHeadingEs);
`${userData.es.chapter} 1, ${userData.es.section} 1.`// Produces: Capítulo
```

Using the code above we can also insert it in our HTML, looking something like this:

```
<h1>`${chapterHeading}` (English)</h1>

<h1>`${chapterHeadingEs}` (Spanish)</h1>
```

The challenge is to dynamically set the current language and use the corresponding entry from the language database. I did a naive pass before finding an external solution that work better.

A more complete example

It's tempting to try and reinvent the wheel (and fail miseable luke I did) it's good to go around and see what's out there.

Andrea Giamarchi's Easy i18n in 10 lines of JavaScript (PoC) provides a more robust idea for how to do translation using template literals. This code has been further developed in a Github Repo. I will stay with the original idea of the post, and leave it to you if you want to use the llibrary.

The first part of this process is to define how we'll handle the i18n templates. This will query the database and, based on the language key, we return the string for the matched language.

It will also set up the default language (en) and an empty internationalization database (`i18n.db`).

```
function i18n(template) {
  for (var
    info = i18n.db[i18n.locale][template.join('\x01')],
    out = [info.t[0]],
    i = 1, length = info.t.length; i < length; i++
  ) out[i] = arguments[1 + info.v[i - 1]] + info.t[i];
  return out.join('');
}
i18n.locale = 'en';
i18n.db = {};
```

The next function creates the database for the translation. We'll use this to populate the database that we'll feed translations to.

```
i18n.set = locale => (tCurrent, ...rCurrent) => {
  const key = tCurrent.join('\x01');
  let db = i18n.db[locale] || (i18n.db[locale] = {});
  db[key] = {
    t: tCurrent.slice(),
    v: rCurrent.map((value, i) => i)
  };
  const config = {
    for: other => (tOther, ...rOther) => {
      db = i18n.db[other] || (i18n.db[other] = {});
      db[key] = {
        t: tOther.slice(),
        v: rOther.map((value, i) => rCurrent.indexOf(value))
      };
      return config;
    }
  };
  return config;
};
```

Andrea provides multiple ways to populate the database. For this example I will

populate it using the `set` method. The example below set a groiup of entries using English as the default language and then using `.for` to identify additional languages and their translation.

```
i18n.set('en') `Hello ${'n'name'}`
  .for('de') 'de'`Hallo ${'n'name'}`for('it') `Cia'it'`Ciao ${'n'name'}`'
```

TO create a database containing translation information for our books could look like this:

```
i18n.set('en') `Chapter ${'n'number'}`
  .for('es') 'es'`Capítulo ${'n'number'}`for('de') `Kap'de'`Kapitel ${'n'
```

We can then use the default language and type the data in English.

```
// default
i18n`Chapter $`Chapter ${'7'73'}`// "Chapter 73"
```

We also have the option of switching languages at run time, continue writing our text in English and see it translated using the database content.

```
// we switch to German but still write in English
i18n.locale = 'de';
i18n`Chapter ${'73'}`;
'de'`Chapter ${'7'73'}`// "Kapitel 73"
i18n`Chapter ${'73'}`;
// Capítulo 73
'73'`Chapter ${'7'73'}`// Capítulo 73
```

This code presents a basic engine that will cover most of our needs, if we're willing to do the data entry ourselves or use the libraries and utilities Andreas present in Github.

> This project is not meant to replace libraries like Globalize, ICU, Unicode CLDR.