Pocket Guides ⚓ Go Make Things

STRINGS, ARRAY, & OBJECT MANIPULATION

with Vanilla JavaScript

CHRIS FERDINANDI

String, Array, & Object Manipulation

By Chris Ferdinandi

Go Make Things, LLC

v2.0.0

Table of Contents

Intro

In this guide, you'll learn:

- How to remove whitespace from a string.
- How to transform text to uppercase, lowercase, and title case.
- How to convert strings to integers.
- How to replace a portion of text with different text.
- How to get a portion of a string.
- How to split a string into an array based on a character.
- How to add items to an array or object.
- How to merge two or more arrays or objects together.

A quick word about browser compatibility

This guide makes heavy use of ECMAScript 5 (more commonly known as ES5) methods and APIs.

That generally means browser support begins with IE9 and above. Each function or technique mentioned in this guide includes specific browser support information, as some do provide further backwards compatibility.

Let's get started…

trim()

`.trim()` is used to remove whitespace from the beginning and end of a string.

```
var text = '  This sentence has some whitespace at the beginning and end of it.  ';
var trimmed = text.trim();
// returns 'This sentence has some whitespace at the beginning and end of it.'
```

Browser Compatibility

Works in all modern browsers, and IE9 and above. The following polyfill[1] can be used to push support back to IE6.

```
if (!String.prototype.trim) {
    String.prototype.trim = function () {
        return this.replace(/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g, '');
    };
}
```

Upper, Lower, and Title Case

How to transform strings into uppercase, lowercase, and title case.

toLowerCase()

Transform all text in a string to lowercase.

```
var text = 'This sentence has some MIXED CASE LeTTeRs in it.'
;
var lower = text.toLowerCase();
// returns 'this sentence has some mixed case letters in it.'
```

Browser Compatibility

Supported in all modern browsers, and at least back to IE6.

toUpperCase()

Transform all text in a string to uppercase.

```
var text = 'This sentence has some MIXED CASE LeTTeRs in it.'
;
var upper = text.toUpperCase();
// returns 'THIS SENTENCE HAS SOME MIXED CASE LETTERS IN IT.'
```

Browser Compatibility

Supported in all modern browsers, and at least back to IE6.

Title Case

While there’s no native JavaScript function for this, you can combine a few methods into a helper function to title case your string.

1. First, we’ll convert our entire string to lowercase.
2. Next, we’ll split the string into an array of words using ` ` as the delimiter.
3. Then, we’ll loop through each word in our array.
4. After that, we’ll capitalize the first letter, and lowercase the rest of the string.
5. Finally, we’ll combine all of the words back together into a string.

```
// https://gist.github.com/SonyaMoisset/aa79f51d78b3963943066
1c03d9b1058#file-title-case-a-sentence-for-loop-wc-js
var toTitleCase = function (str) {
    str = str.toLowerCase().split(' ');
    for (var i = 0; i < str.length; i++) {
        str[i] = str[i].charAt(0).toUpperCase() + str[i].slic
e(1);
    }
    return str.join(' ');
};

// Example
var str = 'HeRe is a MIXED capitization StRiNg.';
var str = toTitleCase(str);
// returns: "Here Is A Mixed Capitization String."
```

Browser Compatibility

Supported in all modern browsers, and IE6 and above.

Converting Strings to Integers

How to convert strings to integers.

parseInt()

Convert a string into an integer (a whole number). The second argument, `10`, is called the `radix`. This is the base number used in mathematical systems. For our use, it should always be `10`.

```
var text = '42px';
var integer = parseInt( text, 10 );
// returns 42
```

Browser Compatibility

Supported in all modern browsers, and at least back to IE6.

parseFloat()

Convert a string into a point number (a number with decimal points).

```
var text = '3.14someRandomStuff';
var pointNum = parseFloat( text );
// returns 3.14
```

Browser Compatibility

Supported in all modern browsers, and at least back to IE6.

Replace Text

How to replace text in a string.

replace()

Replace a portion of text in a string with something else.

```
var text = 'I love Cape Cod potato chips!';
var lays = text.replace( 'Cape Cod', 'Lays' );
var soda = text.replace( 'Cape Cod potato chips', 'soda' );
var extend = text.replace( 'Cape Cod', 'Cape Cod salt and vin
egar' );

// lays: 'I love Lays potato chips!'
// soda: 'I love soda!'
// extend: 'I love Cape Cod salt and vinegar potato chips!'
```

Browser Compatibility

Supported in all modern browsers, and at least back to IE6.

Slicing a String

How to get a portion of a string.

slice()

Get a portion of a string starting (and optionally ending) at a particular character.

The first argument is where to start. Use `0` to include the first character. The second argument is where to end (and is optional).

If either argument is a negative integer, it will start at the end of the string and work backwards.

```
var text = 'Cape Cod potato chips';
var startAtFive = text.slice( 5 );
var startAndEnd = text.slice( 5, 8 );
var sliceFromTheEnd = text.slice( 0, -6 );

// startAtFive: 'Cod potato chips'
// startAndEnd: 'Code'
// sliceFromTheEnd: 'Cape Cod potato'
```

Browser Compatibility

Supported in all modern browsers, and at least back to IE6.

String to Array

How to convert a string to an array based on a character.

split()

Convert a string into an array by splitting it after a specific character (or characters).

The first argument, the `delimiter`, the character or characters to split by. As an optional second argument, you can stop splitting your string after a certain number of delimiter matches have been found.

```
var text = 'Soda, turkey sandwiches, potato chips, chocolate chip cookies';
var menu = text.split( ', ' );
var limitedMenu = text.split( ', ', 2 );

// menu: ["Soda", "turkey sandwiches", "potato chips", "chocolate chip cookies"]
// limitedMenu: ["Soda", "turkey sandwiches"]
```

Browser Compatibility

Supported in all modern browsers, and at least back to IE6.

Add Items to an Array or Object

How to add items to an array or object.

Add items to an array

Use `push()` to add items to an array.

```
var sandwiches = ['turkey', 'tuna', 'blt'];
sandwiches.push('chicken', 'pb&j');
// sandwiches: ['turkey', 'tuna', 'blt', 'chicken', 'pb&j']
```

Browser Compatibility

Works in all modern browsers, and IE6 and above.

Add items to an object

Use the dot notation (`obj.something`) or bracket notation (`obj['something']`) to add key/value pairs to an object.

```
var lunch = {
    sandwich: 'turkey',
    chips: 'cape cod',
    drink: 'soda'
}

// Add items to the object
lunch.alcohol = false;
lunch["dessert"] = 'cookies';

// return: {sandwich: "turkey", chips: "cape cod", drink: "so
da", alcohol: false, dessert: "cookies"}
```

Browser Compatibility

Works in all modern browsers, and at least IE6.

Merge Arrays and Objects

How to merge two or more arrays or objects together.

Merge two or more arrays together

Use `Array.prototype.push.apply()` to merge two or more arrays together. Merges all subsequent arrays into the first.

```
var sandwiches1 = ['turkey', 'tuna', 'blt'];
var sandwiches2 = ['chicken', 'pb&j'];
Array.prototype.push.apply(sandwiches1, sandwiches2);
// sandwiches1: ['turkey', 'tuna', 'blt', 'chicken', 'pb&j']
// sandwiches2: ['chicken', 'pb&j']
```

Browser Compatibility

Works in all modern browsers, and at least IE6.

Merge two or more objects together

`extend` is a helper method I wrote to merge two or more objects together. It works a lot like jQuery's `.extend()` function, except that it returns a new object, preserving all of the original objects and their properties. For deep (or recursive) merges, pass in `true` as the first argument. Otherwise, just pass in your objects.

```
/**
```

```
 * Merge two or more objects. Returns a new object.
 * Set the first argument to `true` for a deep or recursive m
erge
 * @param {Boolean}  deep     If true, do a deep (or recursiv
e) merge [optional]
 * @param {Object}   objects  The objects to merge together
 * @returns {Object}          Merged values of defaults and o
ptions
 */
var extend = function () {

    // Variables
    var extended = {};
    var deep = false;
    var i = 0;
    var length = arguments.length;

    // Check if a deep merge
    if ( Object.prototype.toString.call( arguments[0] ) === '
[object Boolean]' ) {
        deep = arguments[0];
        i++;
    }

    // Merge the object into the extended object
    var merge = function ( obj ) {
        for ( var prop in obj ) {
            if ( Object.prototype.hasOwnProperty.call( obj, p
rop ) ) {
                // If deep merge and property is an object, m
erge properties
                if ( deep && Object.prototype.toString.call(o
bj[prop]) === '[object Object]' ) {
                    extended[prop] = extend( true, extended[p
rop], obj[prop] );
                } else {
                    extended[prop] = obj[prop];
                }
            }
        }
    };

    // Loop through each object and conduct a merge
    for ( ; i < length; i++ ) {
        var obj = arguments[i];
        merge(obj);
    }

    return extended;

};

// Example objects
var object1 = {
    apple: 0,
    banana: { weight: 52, price: 100 },
    cherry: 97
};
var object2 = {
    banana: { price: 200 },
    durian: 100
};
var object3 = {
    apple: 'yum',
    pie: 3.214,
    applePie: true
}

// Create a new object by combining two or more objects
var newObjectShallow = extend( object1, object2, object3 );
var newObjectDeep = extend( true, object1, object2, object3 )
;
```

Browser Compatibility

Works in all modern browsers, and at least IE6.

Putting it all together

To make this all tangible, let’s work on a project together. We’re going to display a list of adoptable dogs for an animal rescue by taking some (fake) API data, manipulating it a bit, and rendering in the markup.

The starter template and complete project code are included in the source code[2] on GitHub.

Getting Setup

I’ve dropped some placeholder code into the template to get us started.

HTML

There’s really not much here. Just a heading, a `<div>` with the `#dogs` ID where we’ll add our list of pets.

```html
<h1>Adoptable Dogs</h1>

<div id="dogs">Fetching our adoptable dogs...</div>
```

CSS

I’ve added just a few lightweight styles to the page: one to make sure our images are responsive, and another to force leading and trailing whitespace.

Whitespace normally collapses automatically (ex. `some  text` displays as `some text`), but for practice purposes, I wanted to force it to display.

```
img {
    height: auto;
    max-width: 100%;
}

p {
    white-space: pre-wrap;
}
```

JavaScript

Since this pocket guide is *not* about Ajax or DOM injection, I added some starter JavaScript to handle that stuff so that you can focus on manipulating strings, arrays, and objects.

First, there's some dummy data that we'll pretend was returned from an API call.

```
var apiData = {
    0: {
        name: 'Rufus',
        breeds: [
            'Lab',
            'German Shepard',
            'Border Collie'
        ],
        age: 'adult',
        size: 'M',
        gender: 'M',
        details: 'No Cats, No Dogs',
        photo: 'img/rufus.jpg',
        description: '      Hail-shot bounty barque  chase guns. Brigantine gibbet haul wind line.  Barque chandler lookout clap of thunder. Transom hogshead trysail league.   '
    },
    ...
};
```

I've included a starter function—`createListing()`—that we'll use to generate each dog listing. We'll pass each dog's data in as an argument (`dog`), and add the relevant data to our template.

Along the way, we'll need to manipulate and transform it to suite our needs.

```
// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2></h2>' +

        '<p><img src=""></p>' +

        '<p>' +
            'Age: <br>' +
            'Size: <br>' +
            'Gender: <br>' +
            'Breeds: ' +
        '</p>' +

        '<strong>Other Details:</strong>' +

        '<p>Description</p>';

    return content;

};
```

I setup a `for` loop to loop through each dog and call that function, passing in the individual dog data as an argument. Then I take the completed markup and inject it into the DOM with `innerHTML`.

```
// Generate a list of adoptable dogs
var dogs = '';
for (dog in apiData) {
    if (apiData.hasOwnProperty(dog)) {
        dogs += createListing(apiData[dog]);
    }
}

// Load list of adoptable dogs into the DOM
var dogList = document.querySelector('#dogs');
dogList.innerHTML = dogs;
```

For this project, we're going to focus on the `createListing()` function.

Adding the dog's name

The first thing we want to do is add the dog's name to the listing. We'll add `dog.name` to our function.

```
// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2>' + dog.name + '</h2>' +

        '<p><img src=""></p>' +

        '<p>' +
            'Age: <br>' +
            'Size: <br>' +
            'Gender: <br>' +
            'Breeds: ' +
        '</p>' +

        '<strong>Other Details:</strong>' +

        '<p>Description</p>';

    return content;

};
```

If you reload the page, you'll notice that some dog's have capitalized names, while others are all lowercase. Let's make all dog names uppercase with the `toUpperCase()` method.

```
// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2>' + dog.name.toUpperCase() + '</h2>' +

        '<p><img src=""></p>' +

        '<p>' +
            'Age: <br>' +
            'Size: <br>' +
            'Gender: <br>' +
            'Breeds: ' +
        '</p>' +

        '<strong>Other Details:</strong>' +

        '<p>Description</p>';

    return content;

};
```

Adding a photo of the dog

Next, let's add each dog's photo to the listing. We'll set the image `src` to `dog.photo`, and also add some `alt` text for non-sighted users.

```
// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2>' + dog.name.toUpperCase() + '</h2>' +

        '<p><img alt="A photo of ' + dog.name + '" src="' + d
og.photo + '"></p>' +

        '<p>' +
            'Age: <br>' +
            'Size: <br>' +
            'Gender: <br>' +
            'Breeds: ' +
        '</p>' +

        '<strong>Other Details:</strong>' +

        '<p>Description</p>';

    return content;

};
```

Adding the dog's age

The dog's age in the API data is lowercase, and we'd like it to be be title case (that is, start with an uppercase letter).

Let’s add our `toTitleCase()` helper function to the script, and pass the dog’s age in.

```
// Convert string to title case
// source: https://gist.github.com/SonyaMoisset/aa79f51d78b39639430661c03d9b1058#file-title-case-a-sentence-for-loop-wc-js
var toTitleCase = function (str) {
    str = str.toLowerCase().split(' ');
    for (var i = 0; i < str.length; i++) {
        str[i] = str[i].charAt(0).toUpperCase() + str[i].slice(1);
    }
    return str.join(' ');
};

// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2>' + dog.name.toUpperCase() + '</h2>' +

        '<p><img alt="A photo of ' + dog.name + '" src="' + dog.photo + '"></p>' +

        '<p>' +
            'Age: ' + toTitleCase(dog.age) + '<br>' +
            'Size: <br>' +
            'Gender: <br>' +
            'Breeds: ' +
        '</p>' +

        '<strong>Other Details:</strong>' +
        '<p>Description</p>';

    return content;

};
```

Looking good so far, but... Kylie Jane and Colt, the puppies, have an age of "Baby." It would be nice if that said "Puppy" instead. We'll use `replace()` to change it.

```
// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2>' + dog.name.toUpperCase() + '</h2>' +

        '<p><img alt="A photo of ' + dog.name + '" src="' + d
og.photo + '"></p>' +

        '<p>' +
            'Age: ' + toTitleCase(dog.age.replace('baby', 'Pu
ppy')) + '<br>' +
            'Size: <br>' +
            'Gender: <br>' +
            'Breeds: ' +
        '</p>' +

        '<strong>Other Details:</strong>' +

        '<p>Description</p>';

    return content;

};
```

Adding the dog's size

We'll add the dog's size to our listing using `dog.size`

```
// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2>' + dog.name.toUpperCase() + '</h2>' +

        '<p><img alt="A photo of ' + dog.name + '" src="' + d
og.photo + '"></p>' +

        '<p>' +
            'Age: ' + toTitleCase(dog.age.replace('baby', 'Pu
ppy')) + '<br>' +
            'Size: ' + dog.size + '<br>' +
            'Gender: <br>' +
            'Breeds: ' +
        '</p>' +

        '<strong>Other Details:</strong>' +

        '<p>Description</p>';

    return content;

};
```

Our API data uses a letter abbreviation for size: `S` for small, `M` for medium, `L` for large, and `XL` for extra large. Let’s convert those to words, again using `replace()`.

```
// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2>' + dog.name.toUpperCase() + '</h2>' +

        '<p><img alt="A photo of ' + dog.name + '" src="' + d
og.photo + '"></p>' +

        '<p>' +
            'Age: ' + toTitleCase(dog.age.replace('baby', 'Pu
ppy')) + '<br>' +
            'Size: ' + dog.size.replace('S', 'Small').replace
('M', 'Medium').replace('L', 'Large').replace('XL', 'Very Lar
ge') + '<br>' +
            'Gender: <br>' +
            'Breeds: ' +
        '</p>' +

        '<strong>Other Details:</strong>' +

        '<p>Description</p>';

    return content;

};
```

That works great... except for extra large dogs. Their size is being rendered as `Very Largearge`. Why is that? It's because of the uppercase `L` in `replace('L', 'Large')`.

What can we do?

If we use lowercase letters for our replacement words, and then use our `toTitleCase()` function to capitalize the finished result, we should be able to avoid this issue.

```
// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2>' + dog.name.toUpperCase() + '</h2>' +

        '<p><img alt="A photo of ' + dog.name + '" src="' + d
og.photo + '"></p>' +

        '<p>' +
            'Age: ' + toTitleCase(dog.age.replace('baby', 'Pu
ppy')) + '<br>' +
            'Size: ' + toTitleCase(dog.size.replace('S', 'sma
ll').replace('M', 'medium').replace('L', 'large').replace('XL
', 'very large')) + '<br>' +
            'Gender: <br>' +
            'Breeds: ' +
        '</p>' +

        '<strong>Other Details:</strong>' +

        '<p>Description</p>';

    return content;

};
```

Almost! Now extra large dogs are displaying as `Xlarge`.

The `replace('L', 'large')` function is catching the `L` in `XL` and changing it. Then, when `replace('XL', 'extra large')` runs next, there’s no `XL` to replace.

Let’s flip those two around.

```
// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2>' + dog.name.toUpperCase() + '</h2>' +

        '<p><img alt="A photo of ' + dog.name + '" src="' + dog.photo + '"></p>' +

        '<p>' +
            'Age: ' + toTitleCase(dog.age.replace('baby', 'Puppy')) + '<br>' +
            'Size: ' + toTitleCase(dog.size.replace('S', 'small').replace('M', 'medium').replace('XL', 'very large').replace('L', 'large')) + '<br>' +
            'Gender: <br>' +
            'Breeds: ' +
        '</p>' +

        '<strong>Other Details:</strong>' +

        '<p>Description</p>';

    return content;

};
```

Perfect!

Adding the dog’s gender

Next, let’s add the dog’s gender. We can do that with `dog.gender`.

```
// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2>' + dog.name.toUpperCase() + '</h2>' +

        '<p><img alt="A photo of ' + dog.name + '" src="' + d
og.photo + '"></p>' +

        '<p>' +
            'Age: ' + toTitleCase(dog.age.replace('baby', 'Pu
ppy')) + '<br>' +
            'Size: ' + toTitleCase(dog.size.replace('S', 'sma
ll').replace('M', 'medium').replace('XL', 'very large').repla
ce('L', 'large')) + '<br>' +
            'Gender: ' + dog.gender + '<br>' +
            'Breeds: ' +
        '</p>' +

        '<strong>Other Details:</strong>' +

        '<p>Description</p>';

    return content;

};
```

Right now, our script displays `M` for males and `F` for females. Let’s use `replace()` to swap that out with some text. This one is a lot more straightforward than the dog’s size.

```
// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2>' + dog.name.toUpperCase() + '</h2>' +

        '<p><img alt="A photo of ' + dog.name + '" src="' + d
og.photo + '"></p>' +

        '<p>' +
            'Age: ' + toTitleCase(dog.age.replace('baby', 'Pu
ppy')) + '<br>' +
            'Size: ' + toTitleCase(dog.size.replace('S', 'sma
ll').replace('M', 'medium').replace('XL', 'very large').repla
ce('L', 'large')) + '<br>' +
            'Gender: ' + dog.gender.replace('M', 'Male').repl
ace('F', 'Female') + '<br>' +
            'Breeds: ' +
        '</p>' +

        '<strong>Other Details:</strong>' +

        '<p>Description</p>';

    return content;

};
```

Adding the dog’s breeds

In our API data, the breeds for each dog are in an array.

```
var apiData = {
    0: {
        name: 'Rufus',
        breeds: [
            'Lab',
            'German Shepard',
            'Border Collie'
        ],
        ...
    },
    ...
};
```

For this one, we'll create a function, `getBreeds()`, to process our array and return a string.

```
// Get a dog's breeds as a string
var getBreeds = function (dog) {
    // Code goes here...
};

// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2>' + dog.name.toUpperCase() + '</h2>' +

        '<p><img alt="A photo of ' + dog.name + '" src="' + d
og.photo + '"></p>' +
        '<p>' +
            'Age: ' + toTitleCase(dog.age.replace('baby', 'Pu
ppy')) + '<br>' +
            'Size: ' + toTitleCase(dog.size.replace('S', 'sma
ll').replace('M', 'medium').replace('XL', 'very large').repla
ce('L', 'large')) + '<br>' +
            'Gender: ' + dog.gender.replace('M', 'Male').repl
ace('F', 'Female') + '<br>' +
            'Breeds: ' + getBreeds(dog) +
        '</p>' +

        '<strong>Other Details:</strong>' +

        '<p>Description</p>';

    return content;

};
```

Now, let’s push each item in the array to a string.

Do that, we’ll create a `breeds` variable and set it to an empty string. Then we’ll loop through our array of breeds, and append our breed name to the end of the `breeds` string.

We’ll also add a `,` to the end of it, separating each of our breeds with a comma.

```
// Get a dog's breeds as a string
var getBreeds = function (dog) {

    // Push each breed to a string
    var breeds = '';
    for (var i = 0; i < dog.breeds.length; i++) {
        breeds += dog.breeds[i] + ', ';
    }

    return breeds;

};
```

That worked great. The one snag: the last breed for each dog ends with a trailing comma.

Fortunately, we can use the `slice()` function to remove it from the end of the string. Remember, `slice()` returns a subset of a string. We'll start with the first character, and remove the last two characters (the space and the ending comma).

```
// Get a dog's breeds as a string
var getBreeds = function (dog) {

    // Push each breed to a string
    var breeds = '';
    for (var i = 0; i < dog.breeds.length; i++) {
        breeds += dog.breeds[i] + ', ';
    }

    // Remove the trailing comma
    breeds = breeds.slice(0, -2);

    return breeds;

};
```

Adding our other details

Each dog comes with a list of additional details. This includes things like, “isn’t good with cats,” or, “is neutered.”

In our API data, it’s a string of items separated by a comma. We’d like to display it as an unordered list, so we want our data in an array so we can loop through it.

Let’s create a new function, `getOtherDetails()`, to handle this for us.

```
// Create a list of other details
var getOtherDetails = function (dog) {
    // Code goes here...
};

// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2>' + dog.name.toUpperCase() + '</h2>' +

        '<p><img alt="A photo of ' + dog.name + '" src="' + d
og.photo + '"></p>' +

        '<p>' +
            'Age: ' + toTitleCase(dog.age.replace('baby', 'Pu
ppy')) + '<br>' +
            'Size: ' + toTitleCase(dog.size.replace('S', 'sma
ll').replace('M', 'medium').replace('XL', 'very large').repla
ce('L', 'large')) + '<br>' +
            'Gender: ' + dog.gender.replace('M', 'Male').repl
ace('F', 'Female') + '<br>' +
            'Breeds: ' + getBreeds(dog) +
        '</p>' +

        '<strong>Other Details:</strong>' + getOtherDetails(d
og) +

        '<p>Description</p>';

    return content;

};
```

The first thing we need to do is convert our string into an array using the `split()` function. We'll pass in a `,` as the delimiter.

```
// Create a list of other details
var getOtherDetails = function (dog) {

    // Convert our string to an array
    var detailsArray = dog.details.split( ', ' );
    console.log(detailsArray);

};
```

If you open up the Console tab in developer tools, you'll see an array of items.

Now let's loop through each item and create our list. Again, we'll set a variable, `details`, to an empty string, and append each detail to the end of it.

When the loop is done, we'll wrap our details in a `<ul>` element and return it.

```
// Create a list of other details
var getOtherDetails = function (dog) {

    // Convert our string to an array
    var detailsArray = dog.details.split( ', ' );

    // Loop through our array and create our list
    var details = '';
    for (var i = 0; i < detailsArray.length; i++) {
        details += '<li>' + detailsArray[i] + '</li>';
    }

    return '<ul>' + details + '</ul>';
};
```

This works great… until you get to Colt. He has no additional details, and we didn't account for that.

A simple `if` statement will fix that. If `dog.details` is an empty string, we'll return a simple message instead.

```
// Create a list of other details
var getOtherDetails = function (dog) {

    // If the array is empty, return a messsage
    if (dog.details === '') {
        return ' No additional details.';
    }

    // Convert our string to an array
    var detailsArray = dog.details.split( ', ' );

    // Loop through our array and create our list
    var details = '';
    for (var i = 0; i < detailsArray.length; i++) {
        details += '<li>' + detailsArray[i] + '</li>';
    }

    return '<ul>' + details + '</ul>';

};
```

Adding a dog’s description

The last thing we need to do is add a description for our dogs. We can do this with `dog.description`.

```
// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2>' + dog.name.toUpperCase() + '</h2>' +

        '<p><img alt="A photo of ' + dog.name + '" src="' + d
og.photo + '"></p>' +

        '<p>' +
            'Age: ' + toTitleCase(dog.age.replace('baby', 'Pu
ppy')) + '<br>' +
            'Size: ' + toTitleCase(dog.size.replace('S', 'sma
ll').replace('M', 'medium').replace('XL', 'very large').repla
ce('L', 'large')) + '<br>' +
            'Gender: ' + dog.gender.replace('M', 'Male').repl
ace('F', 'Female') + '<br>' +
            'Breeds: ' + getBreeds(dog) +
        '</p>' +

        '<strong>Other Details:</strong>' + getOtherDetails(d
og) +

        '<p>' + dog.description + '</p>';

    return content;

};
```

One thing you may notice: the leading spaces before some descriptions. It looks like our API data source didn’t properly trim user submitted data, so we’ll need to do it ourselves with JavaScript.

Note: *In real life, this wouldn't be a problem as browsers collapse leading and trailing spaces. We forced this with* `white-space: pre-wrap;` *in our CSS for learning purposes.*

```
// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2>' + dog.name.toUpperCase() + '</h2>' +

        '<p><img alt="A photo of ' + dog.name + '" src="' + d
og.photo + '"></p>' +

        '<p>' +
            'Age: ' + toTitleCase(dog.age.replace('baby', 'Pu
ppy')) + '<br>' +
            'Size: ' + toTitleCase(dog.size.replace('S', 'sma
ll').replace('M', 'medium').replace('XL', 'very large').repla
ce('L', 'large')) + '<br>' +
            'Gender: ' + dog.gender.replace('M', 'Male').repl
ace('F', 'Female') + '<br>' +
            'Breeds: ' + getBreeds(dog) +
        '</p>' +

        '<strong>Other Details:</strong>' + getOtherDetails(d
og) +

        '<p>' + dog.description.trim() + '</p>';

    return content;

};
```

One last issue: it looks like whoever entered the pet descriptions grew up in the typewriter era and used multiple spaces after periods in some places.

Let’s standardize these at one space after each period using the `replace()` method.

```
// Create the dog listing markup
var createListing = function (dog) {
    var content =
        '<h2>' + dog.name.toUpperCase() + '</h2>' +

        '<p><img alt="A photo of ' + dog.name + '" src="' + dog.photo + '"></p>' +

        '<p>' +
            'Age: ' + toTitleCase(dog.age.replace('baby', 'Puppy')) + '<br>' +
            'Size: ' + toTitleCase(dog.size.replace('S', 'small').replace('M', 'medium').replace('XL', 'very large').replace('L', 'large')) + '<br>' +
            'Gender: ' + dog.gender.replace('M', 'Male').replace('F', 'Female') + '<br>' +
            'Breeds: ' + getBreeds(dog) +
        '</p>' +

        '<strong>Other Details:</strong>' + getOtherDetails(dog) +

        '<p>' + dog.description.replace('  ', ' ').trim() + '</p>';

    return content;

};
```

Congratulations! You just created a dynamic UI by manipulating and sanitizing API data.

About the Author

Hi, I’m Chris Ferdinandi. I help people learn JavaScript.

I love pirates, puppies, and Pixar movies, and live near horse farms in rural Massachusetts. I run Go Make Things with Bailey Puppy, a lab-mix from Tennessee.

You can find me:

- On my website at GoMakeThings.com.
- By email at chris@gomakethings.com.
- On Twitter at @ChrisFerdinandi.

1. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/Trim#Polyfill↩
2. https://github.com/cferdinandi/string-array-object-source-code/↩