What's Happening When We Make AJAX Request to PokeAPI

Traversing JSON

Some URLs have JSON (JavaScript Object Notation) that we can use. JSON is just one of many ways of representing data. Navigate to https://pokeapi.co/api/v2/pokemon/1/ to see JSON data about Bulbasaur. This means that instead of having our own database about Bulbasaur, we can just refer to this link to get more data about this Pokemon. How can we use this JSON data to display Bulbasaur's types (grass and poison), its height, and weight onto our website?

Thanks to jQuery, we have a convenient function *$.get* to do AJAX (which is a way of getting information from another URL without having to reload your own page). If we were to do it using pure JavaScript (which is how *$.get* function in jQuery is implemented), we would have to write a lot of code to make AJAX work for all different types and versions of browsers. Thanks, jQuery!

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Gotta Catch'em All</title>
        <script src="https://code.jquery.com/jquery-3.1.0.min.js"></script>
        <script>
            $(document).ready(function(){
                $.get("https://pokeapi.co/api/v2/pokemon/1/", function(res) {
                    console.log(res);
                }, "json");
            })
        </script>
    </head>
    <body>
        <div id="bulbasaur">
        </div>
    </body>
</html>
```

We are using jQuery to load the JSON data that lives in http://pokeapi.co/api/v2/pokemon/1/ and logging it to the console. We are passing in three arguments to the *$.get* function.

. The URL that we want to load information from (i.e. "http://pokeapi.co/api/v1/pokemon/1/")
. Function you want to run after information was successfully loaded (i.e. function(res) { console.log(res) })
  - *$.get* will automatically pass an argument to this function that holds the information from the URL we requested
  - We can access this argument by having our function take an argument (i.e. *res*)
  - You can name this argument anything you want, you can name it *'data'* for example
. The type of information we are expecting back which in our case is *json.*

You can read more about $.get here.

Traversing JSON

Go ahead and play with the JavaScript Object that is logged onto your console. Chrome has done a great job allowing us to traverse through different parts of the Object by clicking inside. Let's say we want to display Bulbasaur's types (poison, grass), onto our console. How can we do this?

Getting Bulbasaur's Types

Once we click on the Object that is logged in our console, we will see a lot of attributes. One of those attributes is named *'types.'*Chrome has provided us with a nice GUI arrow that specifies that there's more inside. Go ahead and click the arrow and notice that the '*types'* attribute is storing an array of two objects. If we open up each of these objects, we see that they both have a *'name'*attribute. This looks like the data that we want. How can we traverse the JSON to get here?

```
$.get("https://pokeapi.co/api/v2/pokemon/1/", function(res) {
    console.log(res.types[0].type.name);
    console.log(res.types[1].type.name);
}, "json");
```

For the first console.log, we are saying that go to the *'types'* attribute of the object that we got back which can be referred to as *'res.'* Since the *'types'* attribute is holding an array of two objects, we are indexing to the first value to grab the first object. Then, we access that object's name attribute to have 'poison' logged to our console. In the second *console.log*, we are doing the same thing except we are grabbing the second object inside the *res.types array*. Since we know that *res.types* is an array, we can rewrite our code using a for loop. What if the link we are using updates Bulbasaur and adds another type? Using a for loop makes our application more flexible.

```
$.get("https://pokeapi.co/api/v2/pokemon/1/", function(res) {
    for(var i = 0; i < res.types.length; i++) {
        console.log(res.types[i].type.name);
    }
}, "json");
```