Lesson 04.06: JSON (JavaScript Object Notation)

JSON is the syntax of a JS object that has been stringified for the purposes of transmitting the data across a network, most notably, via an HTTP Request on the Internet. An API (Application Programming Interface) typically returns the requested data (such as real-time weather, cryptocurrency prices, etc.) in JSON format.

Given this mini-array of two animal objects.

- access individual animals by array index. The Bison is at index 0.
- each individual animal object has 4 properties.
- use "dot-syntax to "drill in" to get the individual properties

```
const animals = [
    { name: 'American bison',
      class: 'mammal',
      herbivore: true,
      continent: 'North America',
      wildPop: 20000,
      captivity: 400000,
    },
    { name: 'Green Anaconda',
      class: 'reptile',
      herbivore: false,
      continent: 'South America',
      wildPop: 3700000,
      captivity: 50000,
    },
];
```

1. Log 'mammal', 'anaconda', the whole first object and the whole array:

```
console.log(animals[0].class); // mammal
console.log(animals[1].name); // anaconda
console.log(animals[0]); // first object
console.log(animals, typeof(animals)); // array object
```

JSON.stringify(obj) : Turning a JS Object into a String:

If we wanted to transmit the animals array as an HTTP Request --
that is, send it to a webpage across a network -- we would have to "package" the data as a string. Think of mailing a letter (snail mail style). You can't just drop a letter in a mailbox--you have to put it in an envelope. Turning a data object into a string to get it ready to be transmitted is like putting a letter in an envelope. To do this, we pass the object to the JSON.stringify(obj) method JSON.stringify(obj) returns the stringified version, so set the command equal to a variable.

2. Stringify the array of objects, saving the result to a variable:

```
const animalsJSON = JSON.stringify(animals);
console.log(animalsJSON, typeof(animalsJSON)); // string
```

In the Console, compare the original array of objects to the JSON version:

- The array has a length of 2,with individual objects and properties all itemized.
- The JSON from the array is a string -- the individual items and their properites are not accessible
- JSON strings have some characteristics not found in "regular" strings. In a JSON string:
  - all keys are in double quotes
  - all string values ("American Bison") are in double quotes --even if the original object has them in single quotes
  - number and boolean values are NOT in quotes
  - wrapper quotes are around the whole JSON, although these quotes do not show up in the Console output

3. Copy the whole JSON string in the Console, and paste it into your script, inside backticks. Set the whole thing equal to a variable. For ease of reading, break the string up into multiple lines, object-style. The lack of color coding indicates that this is all one big string:

```
let json = `[
    { "name": "American bison",
      "class": "mammal",
      "herbivore": true,
      "continent": "North America",
      "wildPop": 20000,
      "captivity": 400000
    },
    { "name": "anaconda",
      "class": "reptile",
      "herbivore": false,
      "continent": "South America",
      "wildPop": 3700000,
      "captivity": 50000
    }
]`;
```

JSON.parse : Getting a usable object from json string

- The JSON.parse method takes a JSON string as its argument
- JSON.parse parses (breaks up) the JSON string into chunks of data, and returns the result.
- With JSON.parse, we can parse the stringified JSON, which gets us back to our original array of objects

4. Pass the json string to the JSON.parse method, saving the result:

```
const animalsFromJSON = JSON.parse(json);
```

5. Log the result and check the Console. We are back where we started:

```
console.log('animalsFromJSON:\n', animalsFromJSON);
```

Shallow Copying vs. Deep Copying of an Object

If you copy an object, including an array, the copy is not its own, independent entity. Rather, it remains "connected" to the original. If you make a change to the copy, the original changes too. This kind of copy is known as a Shallow Copy.

Shallow Copying: Setting a New Object equal to an Existing Object

- the new object will be "connected to" the original object
- changes made to the new object will affect the original, too

6. Make a Shallow Copy of the Anaconda object.

```
const shallowAnaconda = animals[1];
console.log('shallowAnaconda:\n', shallowAnaconda);
```

7. Make a few changes to the Shallow anaconda object:

```
shallowAnaconda.name = "Shelley the Shallow Anaconda";
shallowAnaconda.specialty = "ambushing prey in shallow water";
```

8. Log anaconda as well as animals:

```
console.log('shallowAnaconda:\n', shallowAnaconda);
console.log('animals:\n', animals);
```

Not only did shallowAnaconda pick up the changes, but so did the anaconda object in the animals array, which was copied. This is what is meant by a "Shallow Copy", and it is not something you would typically want.

Deep Copying: Use JSON.parse(JSON.stringify(obj))

If you pass an object to JSON.stringinfy, and then pass that result to JSON.parse -- that is, if you nest stringify inside parse -- the resulting object will have no "memory of where it came from". Any changes made to the new object will therefore *not* affect the original.

9. Make a Deep Copy of the anaconda from animals using JSON.stringify, nested inside JSON.parse:

```
const deepAnaconda = JSON.parse(JSON.stringify(animals[1]));
```

10. Make a few changes to the deepAnaconda object:

```
deepAnaconda.name = "Denny the Deep Anaconda";
deepAnaconda.specialty = "ambushing prey in deep water";
console.log('deepAnaconda:\n', deepAnaconda);
console.log('animals:\n', animals);
```

The changes to deepAnaconda did NOT affect the original anaconda in animals--which still has the changes imposed on it by shallowAnaconda. And with that we learn that straight-up copying an object by assignment only results in a Shallow Copy. But as a workaround, you can make a Deep Copy of an object by stringifying the object and then immediately parsing the result.

END: Unit 04 Lesson 04.06

NEXT: Unit 05 Lesson 05.01