Objects Cheat Sheet

Here are some notes on what's been covered in this unit. Feel free to copy this and extend it to make your own cheat sheet.

Collections: Objects

Drawbacks of Ordinary Arrays

- A typical array works by referencing elements solely based on their positions, e.g., "the first element, the second element... " etc. But, if the elements are ever rearranged, all of the references to specific elements need to be updated.
- Objects are extremely useful when we want to easily access data. An object generates an enduring relationship between a reference (called a **key**) and the value to which it refers.

Creating Objects

- **Properties** are characteristics associated with an object. In other words, properties tell us about an object.
- Methods are used to represent how people interact with an object in the real world. In other words, **methods** are the actions that can be performed on objects.
- Here's how we would write an object using **literal notation**:

```
var myBicycle = {
  color: "brown",
  model: "DL165",
  make: "Raleigh Competition",
  year: 1976,
  accelerate: function () {
      console.log("Zoom zoom!");
  }
};
```

  - Our object is contained within curly braces `{ }`
  - We are storing our object in a variable, called `myBicycle`
  - We separate each key from its value using a `:`
  - To add a property, we use a property name, such as `color`, `model`, or `make`, followed by a `:`, followed by the corresponding value: `"brown"`, `"DL165"`, or `"Raleigh Competition"`.
  - To add a method, we would use the method name `accelerate`, followed by a `:`, followed by a function (`function () {}`).

- Inside the function, we place any code we want to run when the method is **called**.

Accessing Properties

- When working with objects, we can access values using the object name, followed by a period `.`, followed by the name of the property we want to access:

```
// Store "Raleigh Competition" in the bicycleMake variable.
var bicycleMake = myBicycle.make;
```

Updating and Adding Properties

- To update and add properties, we use the name of the object (in this case, `myBicycle`), followed by a dot `.` followed by the name of the property we want to update or add (in this case, `color` or `year`).
- We then use the assignment operator (`=`), followed by the new value.

```
myBicycle.color = "white";
myBicycle.year = 1977;
```

Accessing Methods

- If we wanted to access — or **call**, our `accelerate` method for our `myBicycle` object — we could do so using the following syntax:

```
// Here we are "calling" the accelerate method.
myBicycle.accelerate();
// => "Zoom zoom!"
```

  - Here, we use the object name `myBicycle`, followed by a period, followed by the method name, followed by parenthesis.

`this`

- In the context of our objects, **`this`** is used in place of the object name to refer to the object.

```
var myBicycle = {
  color: "brown",
  make: "Raleigh Competition",
  getMakeAndColor: function () {
      console.log("My bicycle is a " + this.color + " " + this.make);
  }
};
```

JSON

- **JSON (JavaScript Object Notation)** is a lightweight, text-based data format that's based on JavaScript.
- JSON rules include:
    - Property names must be double-quoted strings (not single quotes).
    - Trailing commas are forbidden.
    - Leading zeros are prohibited.
    - In numbers, a decimal point must be followed by at least one digit.
    - Most characters are allowed in strings; however, certain characters (such as ', ", \, and newline/tab) must be "escaped" with a preceding backslash (\) in order to be read as characters (as opposed to JSON control code).
    - All strings must be double-quoted.
    - No comments allowed!
- If we want to send JavaScript objects from a browser to another application, we can use `JSON.stringify()` to convert our objects into JSON format.

```
favoriteMovie = JSON.stringify(favoriteMovie);
```

- We can use `JSON.parse()` to process a string containing JSON data. The action converts the JSON data into a JavaScript object.

```
favoriteMovie = JSON.parse(favoriteMovie);
```

Creating Objects

1. Create an array called `contacts`. The `contacts` array should contain three objects, one for each contact stored in our book.

```
var contacts = [
    {
        firstName: "John",
        lastName: "Doe",
        phone: "(512) 355-0453",
        email: "johndoe@email.com"
    },
    {
        firstName: "Jane",
        lastName: "Doe",
        phone: "(312) 641-2203",
        email: "janedoe@email.com"
    },
    {
        firstName: "Suzie",
        lastName: "Smith",
        phone: "(415) 604-4219",
        email: "suziesmith@email.com"
    }
];
```

2. Now let's create a `listContacts` function to list our contacts. This function should loop through the `contacts` array and log the first and last name for each contact to the console, e.g. `"John Doe"`. Then call the `listContacts` function.

```
var listContacts = function() {
  for (var i = 0; i < contacts.length; i++) {
    console.log(contacts[i].firstName + ' ' + contacts[i].lastName);
  }
};

listContacts();
```

Methods

1. In the "JavaScript" panel in JS Bin, create an object `poem`

```
var poem = {
```

```
  author: "Robert Frost",
  datePublished: 1916,
  name: "The Road Not Taken",
  famousLine: "Two roads diverged in a yellow wood"

};
```

2. Now add a method `quotePoem` to the `poem` object.

```
var poem = {
  author: "Robert Frost",
  datePublished: 1916,
  name: "The Road Not Taken",
  famousLine: "Two roads diverged in a yellow wood",
  quotePoem: function () {

  }
};
```

3. Inside the method, add a `console.log()` statement, logging the `famousLine` property.

```
var poem = {
  author: "Robert Frost",
  datePublished: 1916,
  name: "The Road Not Taken",
  famousLine: "Two roads diverged in a yellow wood",
  quotePoem: function () {
    console.log(this.famousLine);
  }
};
```

4. Now call the `quotePoem` method.

```
var poem = {
  author: "Robert Frost",
  datePublished: 1916,
  name: "The Road Not Taken",
  famousLine: "Two roads diverged in a yellow wood",
  quotePoem: function () {
    console.log(this.famousLine);
  }
};

poem.quotePoem();
```

JSON

1. The object passes validation. 2. The following line is valid JavaScript, but the object is not written in valid JSON syntax. Use JSONLint to validate the above object; then, update the object to use the correct JSON syntax so it passes the validator.

```
{
    "name": "Eddie Vedder",
    "age": 49
}
```

3. Take a look at the `grungeAlbums` object provided in the JS Bin editor.

- In the "JavaScript" panel in JS Bin, use the `JSON.stringify()`method to turn the `grungeAlbums` JavaScript object into a JSON string, and store it in a variable `grungeAlbumsJSON`.
- Then, use the `JSON.parse()` method to convert `grungeAlbumsJSON`back into a JavaScript object and store it in the variable `grungeAlbumsObject`.
- Log each variable to the console to compare, and also compare to the original object. You'll need to hit the "Run" button in the "Console" panel to see the output from the `console.log()`statements.

```
var grungeAlbumsJSON = JSON.stringify(grungeAlbums);
var grungeAlbumsObject = JSON.parse(grungeAlbumsJSON);

console.log(grungeAlbumsJSON);
console.log(grungeAlbumsObject);
```

4. Now loop through each album in `grungeAlbumsObject`, and use a `console.log()` statement to print out the album name, artist, and units sold.

```
for (var i = 0; i < grungeAlbumsObject.albums.length; i++) {
  var currentAlbum = grungeAlbumsObject.albums[i];
  console.log('Album: ' + currentAlbum.name);
  console.log('Artist: ' + currentAlbum.artist);
  console.log('Units sold: ' + currentAlbum.unitsSold);
}
```