Learning to Code

JavaScript

Student Workbook #4

Version 4.0 Y

DRAFT

author

Dana L. Wyatt, Ph.D.

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Table of Contents

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Module 1

Working with JavaScript Objects

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Section 1–1

JavaScript Objects Literals

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Objects

- **Many programming languages allow programmers to create "objects" in their code**
  - In real life, an object is a "thing" or noun that has many properties associated with it
  - For example: student, course, customer, policy
- **Once you figure out what object you want to model, you have to figure out what properties you need to represent**
  - For example, an Employee might have the following properties:
    * `employeeId`
    * `name`
    * `jobTitle`
    * `payRate`

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

JavaScript Object Literals

- **JavaScript allows you to declare objects literals with values for each property**
  - Properties are written as name and value pairs that are separated by a colon

Example

```
let emp = {
              employeeId: "1",
              name: "Ezra",
              jobTitle: "Web Designer",
              payRate: 45.75
          };
```

  - Spacing, indentation, and line breaks are a matter of preference

Example

```
let emp = { employeeId:"1", name:"Ezra",
            jobTitle:"Web Designer", payRate:45.75 };
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Accessing Object Properties

- **You can access object properties in two ways:**

  `objectName.propertyName`

  `objectName["propertyName"]`

Example

```
let emp1 = {
               employeeId:"1",
               name:"Ezra",
               jobTitle:"Web Designer",
               payRate:45.75
           };

let emp2 = {
               employeeId:"2",
               name:"Elisha",
               jobTitle:"Game Programmer",
               payRate:72.95
           };


console.log("Employee 1: " + emp1.name);
console.log("Employee 2: " + emp2.name);

// OR

console.log("Employee 1: " + emp1["name"]);
console.log("Employee 2: " + emp2["name"]);
```

- **NOTE: dot notation is the most common way of accessing the properties of an object**

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Objects and Functions

- **You can pass objects to functions as parameters**

Example

```
function printEmployeeAndPay(emp) {
   console.log("Name: " + emp.name);
   console.log("Pay: " + emp.payRate);
}


let emp1 = {
               employeeId:"1",
               name:"Ezra",
               jobTitle:"Web Designer",
               payRate:45.75
            };

let emp2 = {
               employeeId:"2",
               name:"Elisha",
               jobTitle:"Game Programmer",
               payRate:72.95
            };

printEmployeeAndPay(emp1);
printEmployeeAndPay(emp2);
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Objects and Functions *cont'd*

- **You can also return objects from functions**
  - This allows you to still return one value from a function by "packaging" several pieces of information together

Example

```
function createPayStub(id, name, payRate, hoursWorked) {
   let grossPay = 0;
   if (hoursWorked <= 40) {
      grossPay = payRate * hoursWorked;
   }
   else {
      grossPay = (40 * payRate) +
                 ((hoursWorked - 40) * 1.5 * payRate);
   }

   let payStub = {
                    employeeId: id,
                    name: name,
                    grossPay: grossPay
                 };
   return payStub;
}


let emp1PayStub =
   createPayStub("1", "Ezra", 45.75, 42);

console.log(emp1PayStub.name + " earned $" +
            emp1PayStub.grossPay.toFixed(2));


let emp2PayStub =
   createPayStub("2", "Elisha", 72.95, 60);

console.log(emp2PayStub.name + " earned $" +
            emp2PayStub.grossPay.toFixed(2));
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Exercises

Create a new folder named `Workbook4`. Create a new folder in the repo named `Mod01`. Then add a subfolder named `ObjectScripts`. The exercises in this section should be placed there.

EXERCISE 1

Create a script that named `label_maker.js`. In it, define a JavaScript object literal with the following properties with sample values of your choice:

```
name
address
city
state
zip
```

Then pass the object to a function named `printContact()`.

You can invoke it using:

```
let myInfo = {
   name: "Pursalane Faye",
   address: "121 Main Street",
   /* other properties not shown */
};
printContact(myInfo);
```

Inside the `printContact()` function, call `console.log()` to print each property formatted like a mailing label.

For example:

```
Pursalane Faye
121 Main Street
Benbrook, Texas 76126
```

EXERCISE 2

Create a script named `product_code.js` that defines a function named `parsePartCode()`. The function takes a string as a parameter formatted as shown below

supplierCode:*productNumber-size*

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Your function will parse the part code (reuse your logic from an earlier exercise) as a JavaScript object. The object it returns will have the following properties:

```
{
   supplierCode: "someValue",
   productNumber: "someValue",
   size: "someValue"
}
```

Call the function and store the return value in an object variable. Then print it out. Try it for several different part codes. For example:

```
let partCode1 = "XYZ:1234-L";
let part1 = parsePartCode(partCode1);
console.log("Supplier: " + part1.supplierCode +
            " Product Number: " + part1.productNumber +
            " Size: " + part1.size);
```

EXERCISE 3

Refactor exercise 1 into `label_maker_with_classes.js`. In it, define a JavaScript class named `Contact` with a constructor function that takes the following parameters:

```
name
address
city
state
zip
```

Then create a method in the class named `printContact()`. You will pass it no parameters -- it will display the data available through the `this` keyword.

You can invoke it using:

```
let myInfo = new Contact("Pursalane Faye", "121 Main Street",
   /* other properties not shown */ );
myInfo.printContact();
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Module 2

Working with Loops and Arrays

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Section 2–1

Loop Basics

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Loops

- **Loops can be used to execute a block of code over and over until some condition is met**

- **There are several types of loops, including:**
  - `while` loop
  - `do/while` loop
  - `for` loop
  - `for-in` loop

- **We will examine several types of loops over the next few pages**
  - However, the `for-in` loop will not be discussed until we talk about arrays

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

`while` Loop

- **The `while` loop executes a block of code for as long as a specified condition is true**
  - If you don't include curly brackets around the block of code in the loop, there can only be one line of code in the `while` statement
  - Best practices says to ALWAYS have brackets

Syntax

```
while (condition) {
   // code to be executed
}
```

Example

```
let num = 1;
let i = 1;

while (i < 5) {
   num = num * 2;
   console.log(num);
   i++;
}
```

OUTPUT

```
2       (i is 1 at the top of the loop / became 2 at the bottom)
4       (i is 2 at the top of the loop / became 3 at the bottom)
8       (i is 3 at the top of the loop / became 4 at the bottom)
16      (i is 4 at the top of the loop / became 5 at the bottom)
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

`do/while` loop

- **The `do/while` is similar to the `while` loop except the condition is checked at the bottom of the loop**
  - This means the loop will execute at least once

Syntax

```
do {
   // code to be executed
} while (condition);
```

Example

```
let num = 1;
let i = 1;

do {
   num = num * 2;
   console.log(num);
   i++;
} while (i < 5)
```

OUTPUT

```
2      (i is 1 at the top of the loop / became 2 at the bottom)
4      (i is 2 at the top of the loop / became 3 at the bottom)
8      (i is 3 at the top of the loop / became 4 at the bottom)
16     (i is 4 at the top of the loop / became 5 at the bottom)
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

`for` loop

- **The `for` loop is typically considered a "counting" loop**

- **It has three parts separated by semicolons:**
  - code that execute before the loop begins
  - a condition that must be true for the loop to keep executing
  - code that runs at the bottom of each iteration

Syntax

```
for (part 1; part 2; part 3) {
   // code to be executed
}
```

Example

```
let num = 1;
let i;

for (i = 0; i < 5; i++) {
   num = num * 2;
   console.log(num);
}

OUTPUT
2       (i is 0 at the top of the loop)
4       (i is 1 at the top of the loop)
8       (i is 2 at the top of the loop)
16      (i is 3 at the top of the loop)
32      (i is 4 at the top of the loop)
```

  - In this example, we started `i` at `0` and added `1` to it each time through the loop but that is not a requirement

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

`for` loop *cont'd*

- **The loop variable can be scoped to the for by defining it in the 'part 1' portion of the loop**

Example

```
let num = 1;

for (let i = 0; i < 5; i++) {
   num = num * 2;
   console.log(num);
}
```

OUTPUT

```
2       (i is 0 at the top of the loop)
4       (i is 1 at the top of the loop)
8       (i is 2 at the top of the loop)
16      (i is 3 at the top of the loop)
32      (i is 4 at the top of the loop)
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Breaking Out of a Loop

- **In any of the loops we've seen thus far, you can use a `break` statement to exit the loop**

Example

```
let num = 1;
let i = 1;

while (i < 100) {
   num = num * 2;
   if (num >= 100) break;
   i++;
}
```

  - It can be useful when you are searching a list for something and you find it!

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Exercises

Create a new folder in this workbook named `Mod02`. Then add a subfolder named `LoopScripts`. The exercises in this section should be placed there.

EXERCISE 1

Define a script named `for_loops.js`. Use a `for` loop to print out the phrase "I love loops" 7 times.

EXERCISE 2

Define a script named `while_loops.js`. Use a `while` loop to print out the phrase "I love loops" 7 times.

Loops will be more fun in a few minutes when we talk about arrays!

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Section 2–2

Arrays

DRAFT

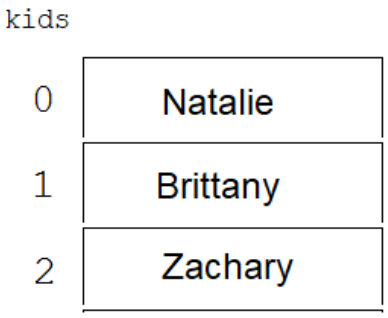
Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Arrays

- **A JavaScript array is used to store multiple values in a single variable**

Example

```
let kids = ["Natalie", "Brittany", "Zachary"];
```



- **To access an element in an array, you use a subscript representing the item's position in the array**
  - Subscripts in JavaScript are **0-based**

Example

```
let kids = ["Natalie", "Brittany", "Zachary"];

let oldest = kids[0];
let middle = kids[1];
let youngest = kids[2];
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Arrays *cont'd*

- **You can also use a variable as a subscript**
  - This is where loops get interesting!

Example

```
let kids = ["Natalie", "Brittany", "Zachary"];

// each time thru the loop kids[i] references a different
// element in the array

for(let i = 0; i < 3; i++) {
   console.log(kids[i]);
}
```

- **You can use the `length` property to get the number of elements in an array**
  - This keeps you from hard coding the size of the array and then getting in a jam because it changes for some reason

Example

```
let kids = ["Natalie", "Brittany", "Zachary"];

for(let i = 0; i < kids.length; i++) {
   console.log(kids[i]);
}
```

- **Best practice: Store the length of an array in a variable if you use it in a loop**
  - This keeps the JavaScript engine from having to recalculate the length each time through the array

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Arrays *cont'd*

Example

```
let kids = ["Natalie", "Brittany", "Zachary"];

let numKids = kids.length;
for(let i = 0; i < numKids; i++) {
   console.log(kids[i]);
}
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

What Can Arrays Hold?

- **JavaScript arrays can store any type of data**

Example

```
// an array that stores all numbers
let mileAgeLog = [313, 328, 349, 287, 301];

// an array that stores all dates
let importantDates = [
                        new Date(1958, 8, 5),
                        new Date(1976, 4, 30),
                        new Date(2009, 10, 10)
                      ];

// an array that stores objects
let menu = [
              {item: "Hamburger", price: 6.95},
              {item: "Cheeseburger", price: 7.95},
              {item: "Hot dog", price: 4.95}
           ];
```

- **Arrays can even store a collection of different data types**

Example

```
let lunch = ["Steak fajitas", 9.95, "Sweet Tea", 2.79];
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Arrays and Functions

- **In JavaScript, you can pass an array to a function**
  - You can also return an array from a function

Example

```
// returns an array of names
function getKids() {
   let kids = ["Natalie", "Brittany", "Zachary"];
   return kids;
}

// displays data in an array of names
function displayKids(kids) {
   let numKids = kids.length;
   for(let i = 0; i < numKids; i++) {
      console.log(kids[i]);
   }
}


let ourKids = getKids();  // returns an array
displayKids(ourKids);     // pass an array
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Looping Though an Array of Objects

- **When you loop through an array of objects, you must use the subscript after the array name and then the property name after the subscript**

Example

```
function getMealCost(orders) {
   let sum = 0;

   let numOrders = orders.length;
   for(let i = 0; i < numOrders; i++) {
      sum += orders[i].price;
   }

   return sum;
}


let myOrder = [
               {item: "Chicken Tacos", price: 8.95},
               {item: "Guacamole", price: 2.85},
               {item: "Sweet Tea", price: 2.75}
];

let yourOrder = [
                 {item: "Hamburger", price: 6.95},
                 {item: "Fries", price: 2.25},
                 {item: "Sweet Tea", price: 2.75},
                 {item: "Fried Apple Pie", price: 4.95}
];

let mealCost = getMealCost(myOrder);
let totalWithTip = mealCost * 1.2;
console.log("My meal costs " + totalWithTip.toFixed(2));

mealCost = getMealCost(yourOrder);
totalWithTip = mealCost * 1.2;
console.log("Your meal costs " + totalWithTip.toFixed(2));
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Exercises

Add a subfolder to `Mod02` named `ArrayScripts`. The exercises in this section should be placed there.

EXERCISE 1

Write a script named `my_family.js` that declares an array with 4 names in it. Loop thru and print them out.

EXERCISE 2

Write a script named `avg_scores.js` that declares two arrays of exam scores.

```
let myScores = [92, 98, 84, 76, 89, 99, 100];
let yourScores = [82, 98, 94, 88, 92, 100, 100];
```

Now, create a function named `getAverage()` to find the average score in that array. (To find an average, loop through and add up all the numbers in the array and then divide by the length of the array) Return the average.

Call your `getAverage()` function and pass it `myScores`. Catch the return value and display it as my average. Repeat with your scores.

EXERCISE 3

Write a script named `foods.js` that declares an array that contains objects you ordered the last time you ate out. For example,

```
let lunch = [
              {item: "Steak Fajitas", price: 9.95},
              {item: "Chips and Guacamole", price: 5.25},
              {item: "Sweet Tea", price: 2.79}
            ];
```

Write code to loop through the array and add up the price of everything you ate and print it out as a subtotal.

Also display the tax on that total (assume 8, the tip on that total (assume 18%), and the total due.

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Expanding an Array

- **You add elements to an array after it has been built by assigning a value to an index number outside the bounds of the array**
  - Any values that are unassigned will hold `undefined`

Example

```
let kids = ["Natalie", "Brittany", "Zachary"];
kids[3] = "Brandon";
kids[5] = "Christina";

console.log(kids);
```

OUTPUT

```
["Natalie",
 "Brittany",
 "Zachary",
 "Brandon",
 ,
 "Christina"]
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Searching an Array

- **There are two functions that make searching some arrays easy**

- **`indexOf()` searches the array for an element and returns its position**
  - It returns -1 if the item is not found

Example

This code searches the list from the beginning

```
let teams = ["Red Sox", "Rangers", "Blue Jays",
             "Astros", "White Sox", "Rangers"];

let index = teams.indexOf("Rangers");     // returns 1
if (index == -1)
   console.log("Item not found");
else
   console.log("Item at position: " + index);
```

  - If you pass a start position, `indexOf()` searches from that position rather than the start of the array

Example

This code searches the list from the position 3

```
let teams = ["Red Sox", "Rangers", "Blue Jays",
             "Astros", "White Sox", "Rangers"];

let index = teams.indexOf("Rangers", 3);
if (index == -1)
   console.log("Item not found");
else
   console.log("Item at position: " + index);
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Searching an Array *cont'd*

- **`lastIndexOf()` it searches the array for an element starting at the end and returns its position**
  - It also returns -1 if the item is not found

Example

This code searches the list from the position 3

```
let teams = ["Red Sox", "Rangers", "Blue Jays",
             "Astros", "White Sox", "Rangers"];

let firstIndex = teams.indexOf("Rangers");         // returns 1
let lastIndex = teams.lastIndexOf("Rangers");      // returns 5
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

inding Subsets of Arrays

- **One of the things you do often is search an array to find a collection of elements that match a specific condition**

Example

```
let menu = [
    {id: 1, item: "Tacos", category: "Meal", price: 12.29},
    {id: 2, item: "Burger", category: "Meal", price: 7.29},
    {id: 3, item: "Salad", category: "Meal", price: 8.29},
    {id: 4, item: "Ice tea", category: "Drink", price: 2.19},
    {id: 5, item: "Coke", category: "Drink", price: 2.29},
    ...
];


function getMenuItemsInCategory(menu, category) {
   let matching = [];

   let numItems = menu.length;
   for(let i = 0; i < numItems; i++) {
      if (menu[i].category == category) {
         matching.push(menu[i]);
      }
   }

   return matching;
}


// show all the drinks
let drinks = getMenuItemsInCategory(menu, "Drink");
let numDrinks = drinks.length;
for(let i = 0; i < numDrinks; i++) {
   console.log(drinks[i].item +
               " $" + drinks[i].price.toFixed(2));
}
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Exercises

EXERCISE 1

Create a script named `course_search.js`. Add a course array to it that resembles:

```
let data = [
    {
        CourseId: "19SUM100",
        Title: "Introduction to HTML/CSS/Git",
        Location: "Classroom 7",
        StartDate: "07/08/19",
        Fee: "100.00",
    },
    {
        CourseId: "19SUM200",
        Title: "Introduction to JavaScript",
        Location: "Classroom 7",
        StartDate: "07/22/19",
        Fee: "350.00",
    },
    {
        CourseId: "19SUM300",
        Title: "Introduction to Node.JS and Express",
        Location: "Classroom 7",
        StartDate: "09/09/19",
        Fee: "50.00",
    },
    {
        CourseId: "19SUM400",
        Title: "Introduction to SQL and Databases",
        Location: "Classroom 7",
        StartDate: "09/16/19",
        Fee: "50.00",
    },
    {
        CourseId: "19SUM500",
        Title: "Introduction to Angular",
        Location: "Classroom 7",
        StartDate: "09/23/19",
        Fee: "50.00",
    }
];
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Exercises *cont'd*

Write 4 functions that are passed the courses array and search it using loops and `if` statements and display the answer to the following questions:

```
// When does the 19SUM200 course start?

// What is the title of the 19SUM500 course?

// What are the titles of the courses that cost $50 or less?

// What classes meet in "Classroom 1"?
```

Define the array at the top of the script.

Don't forget to call the functions and pass the array to them!

EXERCISE 2

Create a script named `cheap_candy.js` that defines an array called `products`. It should contain the following items:

```
let products = [
   { product: "Gummy Worms", price: 5.79 },
   { product: "Plain M&Ms", price: 2.89 },
   { product: "Peanut M&Ms", price: 2.89 },
   { product: "Swedish Fish", price: 3.79 },

   // TODO:  fill the array with 10 candies of various
   //        price ranges
];
```

Write 3 functions that are passed the products array, search it using loops and `if` statements, and return a subset of products or a boolean that answer to the following questions:

```
// What candy costs less that $4.00?

// What candy has "M&M" its name?

// Do we carry "Swedish Fish"?
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Define the array at the top of the script.

Call the functions and pass the array to them. Catch whatever value they return and then display those value(s) to answer the questions!

(Challenge) EXERCISE 3

Create a script named `actors.js` that defines an array called `academyMembers`. It should contain the following items:

```
let academyMembers = [
    {
      memID: 101,
      name: "Bob Brown",
      films: ["Bob I", "Bob II", "Bob III", "Bob IV"]
    },
    {
      memID: 142,
      name: "Sallie Smith",
      films: ["A Good Day", "A Better Day"]
    },
    {
      memID: 187,
      name: "Fred Flanders",
      films: ["A", "BB", "CCC", "DDDD"]
    },
    {
      memID: 203,
      name: "Bobbie Boots",
      films: ["Walking Boots", "Hiking Boots",
               "Cowboy Boots"]
    },
];
```

Notice this array has a TRICK! Each object in the array has a property that is itself an array! This will mean you need two subscripts sometimes:

```
academyMemebers[1].name         -- returns Sallie Smith
academyMemebers[2].films[1]     -- returns Hiking Boots
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Exercises *cont'd*

Write 4 functions that are passed the `academyMembers` array and search it using loops and `if` statements, and return the answer(s) to the following questions::

```
// Who is the Academy Member whose ID is 187?

// Which Academy Members have been in at least 3 films?

// Which Academy Members name starts with "Bob"?

// HARD: Which Academy Members have been in a film
// that starts with "A"
```

(Bonus) EXERCISE 4

Create a script named `vehicle_search.js` that defines an array called `vehicles`. It should contain the following items:

```
let vehicles = [
    {
        color: "Silver",
        type: "Minivan",
        registrationState: "CA",
        licenseNo: "ABC-101",
        registrationExpires: new Date("3-10-2022"),
        capacity: 7
    },
    {
        color: "Red",
        type: "Pickup Truck",
        registrationState: "TX",
        licenseNo: "A1D-2NC",
        registrationExpires: new Date("8-31-2021"),
        capacity: 3
    },
    {
        color: "White",
        type: "Pickup Truck",
        registrationState: "TX",
        licenseNo: "A22-X00",
        registrationExpires: new Date("9-31-2021"),
        capacity: 6
    },
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

```
    {
        color: "Red",
        type: "Car",
        registrationState: "CA",
        licenseNo: "ABC-222",
        registrationExpires: new Date("12-10-2021"),
        capacity: 5
    },
    {
        color: "Black",
        type: "SUV",
        registrationState: "CA",
        licenseNo: "EEE-222",
        registrationExpires: new Date("12-101-2021"),
        capacity: 7
    },
    {
        color: "Red",
        type: "SUV",
        registrationState: "TX",
        licenseNo: "ZZ2-101",
        registrationExpires: new Date("5-30-2021"),
        capacity: 5
    },
    {
        color: "White",
        type: "Pickup Truck",
        registrationState: "TX",
        licenseNo: "CAC-7YT",
        registrationExpires: new Date("1-31-2022"),
        capacity: 5
    },
    {
        color: "White",
        type: "Pickup Truck",
        registrationState: "CA",
        licenseNo: "123-ABC",
        registrationExpires: new Date("3-31-2021"),
        capacity: 5
    }
];
```

Write code that searches the array to find answers to the following. Each search should be in its own function.

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Exercises *cont'd*

```
// What are all the RED vehicles?

// what are all the vehicles whose registrations have expired?

// what are the vehicles that hold at least 6 people?

// Which vehicles have license plates that end with "222"?
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Module 3

Working with Forms - Part 3

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Section 3–1

Working with Lists

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Working with Lists

- **The `<select>` element in HTML creates a list**
  - It is a dropdown list if a `size` attribute isn't specified

Example

```
<select id="statesList" name="states">
  <option value="CO">Colorado</option>
  <option value="ME">Maine</option>
  <option value="TX">Texas</option>
  <option value="WA">Washington</option>
</select>
```

  - It is a listbox if a `size` attribute is specified

Example

```
<select id="statesList" name="states" size="5">
  ...
</select>
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Loading <select> from an Array

- **You can load a <select> from an array when the onload event fires**

Example

HTML

```
<select id="statesList">
</select>
```

JavaScript

```
window.onload = function() {

   // load the dropdown list
   let states = ["Alabama", "Alaska", "Arizona",  ... ];
   let abbrev = ["AL", "AK", "AZ",  ... ];

   const statesList = document.getElementById("statesList");

   let length = states.length;
   for (let i = 0; i < length; i++) {
      let theOption = document.createElement("option");
      theOption.textContent = states[i];
      theOption.value = abbrev[i];
      statesList.appendChild(theOption);
   }

   // other stuff
   ...

};
```

- **Rather than creating and adding the option in 4 lines of code, it can be reduced to 2 lines of code**

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Loading <select> from an Array *cont'd*

Example

```
window.onload = function() {

   // load the dropdown list
   let states = ["Alabama", "Alaska", "Arizona",  ... ];
   let abbrev = ["AL", "AK", "AZ",  ... ];

   const statesList = document.getElementById("statesList");

   let length = states.length;
   for (let i = 0; i < length; i++) {
      let theOption = new Option(states[i], abbrev[i]);
      statesList.appendChild(theOption);
   }

   // other stuff
   ...

};
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Exercises

Create a new folder in the workbook named `Mod03`.

EXERCISE 1

Add a subfolder named `Football`. The exercise in this section should be placed there.

Design a web page that:

1. provides the user with the ability to select a football team from a dropdown list
2. allows the user to click on a button after selecting the team
3. provides a paragraph below the button where we can place text using JavaScript

In this exercise, we will load the dropdown list programmatically. We will save other activities for a future lab.

In your JavaScript file for the page, add the following array at the top under the "use strict".

```
let teams = [
  {code:"DAL", name:"Dallas Cowboys", plays:"Arlington, TX"},
  {code:"DEN", name:"Denver Broncos", plays:"Denver, CO"},
  {code:"HOU", name:"Houston Texans", plays:"Houston, TX"},
  {code:"KAN", name:"Kansas City Chiefs",
            plays:"Kansas City, MO"},
];
```

Write code to handle the window object's `onload` event. In that load event handler, write code to load the football teams into the `select`. Use the team's name for the option text and the team's code for the option value. For example:

```
text:  Dallas Cowboys      value: DAL
```

Test your page.

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Determining the Option Selected

- **You can use JavaScript to interact with the list and to determine user choices**

- **It is very easy to determine the `value` of the selected option**

Example

```
const statesList = document.getElementById("statesList");

let selectedValue = statesList.value;
    // returns TX if Texas is selected in the previous example
```

- **To determine the index number of the selected option, you can use the `selectedIndex` property**
  - It is -1 if nothing is selected

Example

```
const statesList = document.getElementById("statesList");

if (statesList.selectedIndex >= 0) {
   alert("You selected # " + statesList.selectedIndex);
}
```

- **You can use the selected index to look up the actual item in the `select` element's `options` collection**

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Determining the Option Selected *cont'd*

Example

```
const statesList = document.getElementById("statesList");

if (statesList.selectedIndex >= 0) {
   let text =
      statesList.options[statesList.selectedIndex].text;

   let value = statesList.value;

   alert("Selected: " + text + "\nValue: " + value);
}
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Programmatically Selecting/De-Selecting Options

- **You can programmatically select an option in the dropdown by setting the value property to the one you want selected**

Example

HTML

```
<select id="statesList" name="states">
  <option value="CO">Colorado</option>
  <option value="TX">Texas</option>
  <option value="WA">Washington</option>
</select>
```

JavaScript

```
window.onload = function() {
   const statesList = document.getElementById("statesList");
   statesList.value = "TX";  // selects Texas
}
```

- **You can programmatically deselect all items in the `<select>` by setting `selectedIndex` to -1**

Example

```
const statesList = document.getElementById("statesList");
statesList.selectedIndex = -1;
```

- **You can also do it by setting the `value` to `null`**

Example

```
const statesList = document.getElementById("statesList");
statesList.value = null;
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Removing an Option from a <select> List

- **You can programmatically remove an item from a <select> list**
  - You can also call the `remove()` method

Example

```
const statesList = document.getElementById("statesList");

let itemToDelete = "ME";

let length = statesList.options.length;
for (let i = 0; i < length; i++) {
   if (statesList.options[i].value == itemToDelete) {
      statesList.remove(i);
      break;
   }
}
```

  - Assign the option you want to remove the value `null`

Example

```
const statesList = document.getElementById("statesList");

let itemToDelete = "WA";

let length = statesList.options.length;
for (let i = 0; i < length; i++) {
   if (statesList.options[i].value == itemToDelete) {
      statesList.options[i] = null;
      break;
   }
}
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Clearing All Options in a `<select>` List

- **You can clear all options in a `<select>` list by setting the length of the `options` array to 0**

Example

```
<select id="statesList" name="states">
  <option value="CO">Colorado</option>
  <option value="ME">Maine</option>
  <option value="TX">Texas</option>
  <option value="WA">Washington</option>
</select>


// When some event occurs, you can write:

const statesList = document.getElementById("statesList");
statesList.options.length = 0;
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

onchange Event

- **The onchange event occurs when the value of a <select> element has been changed**
  - You can handle it to provide immediate action upon the change

Example

```
window.onload = function() {
   const statesList = document.getElementById("statesList");
   statesList.onchange = onStatesSelectionChanged;

   // other things
   ...
};



function onStatesSelectionChanged() {
   const statesList = document.getElementById("statesList");
   let selectedValue = statesList.value;

   // now do something with selectedValue
}
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Exercises

EXERCISE 1

Continue working in the football program.

Now, write code to handle the button's `click` event. In that event handler, determine which team the user selected and place a message in the paragraph. For example, the message could say:

```
You selected the Dallas Cowboys (DAL) who play in Arlington, TX
```

Test your page.

(Optional) EXERCISE 2

Add a subfolder named `MenuUI`. The exercise in this section should be placed there.

Design a web page that allows a user to select a category of menu items. They page responds by displaying them in a listbox

1. provides the user with a dropdown that contains "Drink", "Meal" and "Dessert" (loaded with options in the HTML)
2. provide a listbox that will show the items that match that category on the menu of a particular restaurant

When the program detects the `onchange` event for the category dropdown list, clear the items in the listbox and then re-load them with the matching items from the selected menu.

Suggestion: place text that combines name and price in a format like "Sweet Team ($2.75)" into the list box that displays the results.

Test your page.

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Module 4

Odds and Ends

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Section 4–1

Minification

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Minification

- **Minification is term applied to reducing the size of your JavaScript code, CSS files, and HTML markup**
  - This helps reduce bandwidth usage and load times for web pages
- **Why minify HTML, CSS, and JavaScript?**
  - Because we use spacing, comments and well-named variables to make code and markup readable
  - Although this makes developing and working on the code easier, it increases network traffic and makes the browser take longer to parse and load the page
- **To minify JS, CSS and HTML files, you must perform several tasks including:**
  - remove comments
  - remove extra whitespace characters
  - make long variable names short
- **If you do a good job with minification, you can reduce the byte count of your code and markup by 40-50%**
- **However, minified code is almost unreadable**
  - This means you need to keep a copy of your original files before minification to use during coding, and deploy the minimized files to production

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Tools to Help with Minification

- **Performing minification can be cumbersome**
  - In fact, manual minification is considered a bad practice
- **Instead, most developers use a third-part tool for minification**
- **Trying to keep up with the tools that are popular is tricky**
  - The following article appeared on the web last year and lists 10 popular tools for 2019

```
https://blog.bitsrc.io/10-javascript-compression-tools-
and-libraries-for-2019-f141a0b15414
```

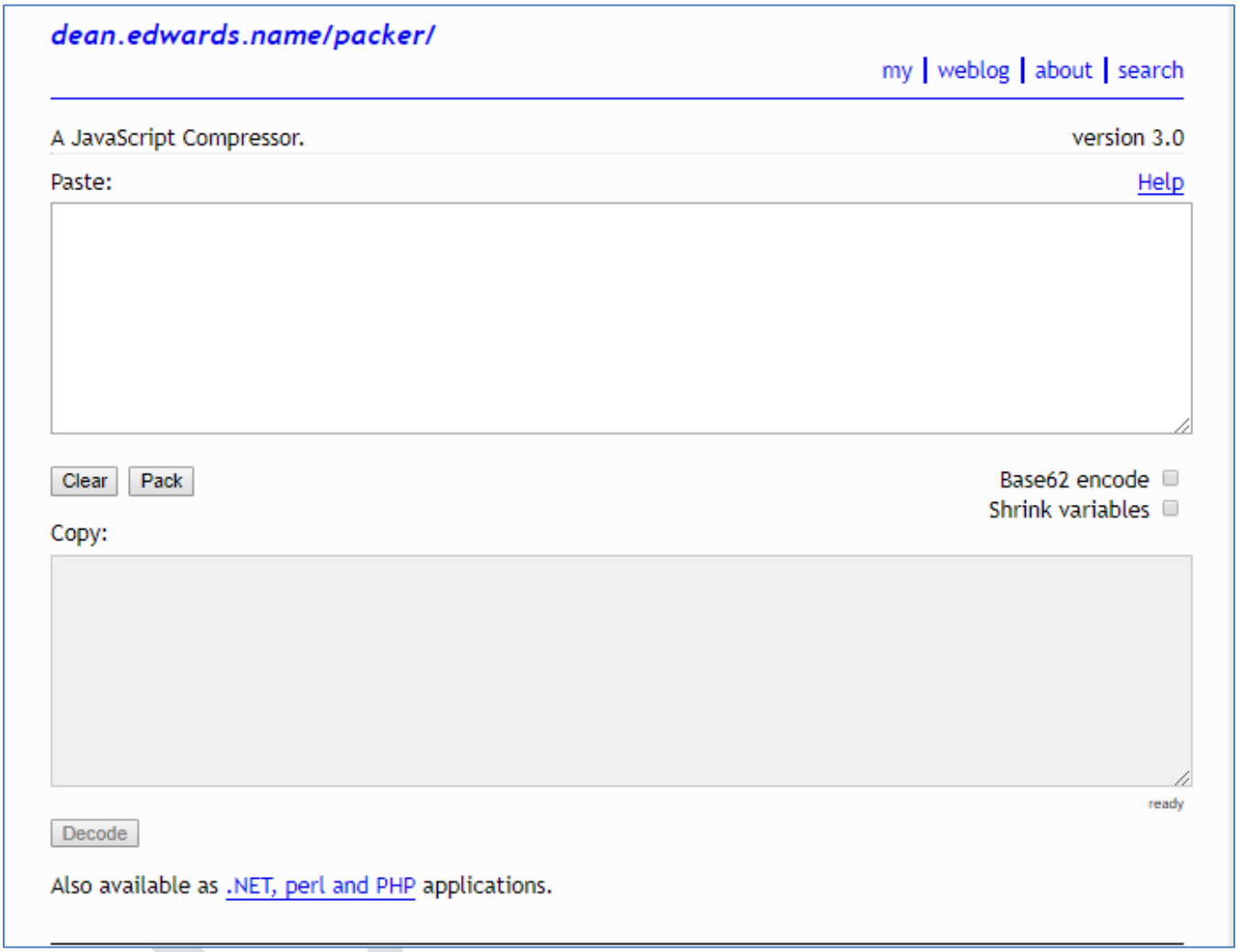
- **Popular options include:**
  - Packer - which is a free online tool
  - Google Closure Compiler - which is available on github
  - Grunt - which is a tool used to automatically perform frequent tasks such as minification, compilation, unit testing and more
    - It is available using the Node.js package manager tool
  - Gulp - which is an slightly newer alternative to Grunt
    - It is also available using the Node.js package manager tool
- **For this exercise, we will use Packer**
  - You can find it at `http://dean.edwards.name/packer/`

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Minifying using Packer

- **When you navigate to the Packer web site, you will see the following page**



  - Paste your script code into the top box
  - Check the Shrink variables checkbox and then click Pack
  - Finally, copy the packed code from the bottom box into a new script file and use that for deployment
    - Typically, a minified file has a name that includes `.min`

```
myscript.js

myscript.min.js
```

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Example: Minified Script

Example

ORIGINAL SCRIPT

```
function generateName(firstName, lastName) {
   let fullName = lastName + ", " + firstName;
   return fullName;
}
```

MINIFIED SCRIPT

```
function generateName(a,b){let c=b+", "+a;return c}
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

JavaScript Libraries and Minification

- **Minification is a standard practice for all web developers**
- **Even major JavaScript library libraries ship in both developer version and minified versions**
  - For example, Bootstrap, JQuery, AngularJS, etc.
- **Typically, minified versions of these popular JavaScript libraries end their file name with the `min.js` suffix**

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Exercises

Create a new folder in the repo named `Mod04`. Add a subfolder named `Financial_Calculators_Reimagined`. The exercise in this section should be placed there.

(Optional) EXERCISE 1

Copy the all files from your week 1 workshop and put them in this new folder.

Copy the code from `mortgage.js` and paste it into the Packer minification tool.

Create a new script file named `mortgage.min.js` and copy the code from Packer there. Now change the `<script>` in the page to reference this minified file instead of the original.

Re-test the page. It should work as before.

View `https://www.devsaran.com/blog/10-best-javascript-minifying-tools`. Try a different minifier. What did it do differently?

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Grunt and Gulp

- **Although using Packer as we did just a few minutes ago was easy, it would be quite painful to do this for many `.css` files and a dozen `.js` files**

- **A tool like Grunt or Gulp automates the minification process**

- **It requires some time to set up, but once that is done, the minification process is automatic**

- **At your leisure, take some time to read about these tools**

  `https://msdn.microsoft.com/en-us/magazine/mt595751.aspx`

  `https://www.hongkiat.com/blog/automate-tasks-vs-code/`

  `https://love2dev.com/blog/using-gruntjs-to-bundle-and-minify-javascript-and-css/`

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Section 4–2

JavaScript and Truthy/Falsy Values

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Truthy/Falsy Values

- **JavaScript uses the terms truthy and falsy values to describe how conditionals will work**
  - Truthy values resolve to true in a Boolean context
  - Falsy values resolve to false in a Boolean context

- **Many of the things we've seen so far that resolve to true or false are easy to understand**

Example

```
let x = 19;
let y = 19;

if (x == y) {          // this expression is true

}
```

Example

```
let x = 19;
let y = 20;

if (x == y) {          // this expression is false

}
```

Example

```
let x = "Cat";
let y = "cat";

if (x != y) {          // this expression is true

}
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Truthy/Falsy Values *cont'd*

- **But sometimes, different values compared with == equate to true because JavaScript converts each to a string representation before comparison**
  - Similar rules apply to **!=**

Example

All of the following are true

```
if (1 == '1') { ... }
if (1 == [1]) { ... }
if ('1' == [1]) { ... }
```

Example

All of the following are false

```
if (1 != '1') { ... }
if (1 != [1]) { ... }
```

  - In JavaScript, the operator **==** is used for loose or abstract equality

- **JavaScript treats the following values as falsy**
  - `false` (of course!)
  - `0`
  - `null`
  - `undefined`
  - empty string
  - `NaN`

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Truthy/Falsy Values *cont'd*

- **Everything else is truthy, including:**
  - `true` (of course!)
  - `"0"` (a string containing a single zero)
  - `"false"` (a string containing the text "false")
  - `[ ]` (an empty array)
  - `{ }` (an empty object)
  - `function(){}` (an "empty" function)

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Strict Equality

- **JavaScript defines the === operator for strict equality**
  - Strict equality is much easier to understand because the value types must match

Example

```
let x = 1;
let y = "1";

if (x == y) { ... }        // returns true

if (x === y) { ... }       // returns false
```

- **Similarly, JavaScript defines the !== operator for strict inequality**
  - **!==** returns the opposite of what **===** would return

Example

```
let x = 1;
let y = "1";

if (x !== y) { ... }       // returns true
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Exercises

EXERCISE 1

Go the website below:

```
https://www.sitepoint.com/javascript-truthy-falsy/
```

Examine the table below "Loose Equality Comparisons With ==="

Contrast that with the table below "Strict Equality Comparisons With ==="

Be careful as you make comparisons in the future! You've probably just been lucky so far that nothing went wrong!

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Section 4–3

Introducing JSON

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

JSON

- **JSON (JavaScript Object Notation) is a standard format for representing text-based structured data based**
  - Objects are stored within curly braces
  - Data is in property/value pairs
- **It is based on JavaScript object syntax**
  - However, the property names must be double-quoted strings

Example

```
{
   "employeeId" : "1",
   "name" : "Ezra",
   "jobTitle" : "Web Designer",
   "payRate" : 45.75
};
```

- **JSON values must be one of the following data types:**
  - a string
  - a number
  - a Boolean
  - an array
  - `null`
  - another JSON object

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

JSON *cont'd*

Example

```
{
   "department" : "Computer Science",
   "location" : "Zachary Bldg Room 301",
   "faculty" : [
      {
        "name" : "Dr. Sallie",
        "rank" : "Professor",
        "phone" : "555-1212"
      },
      {
        "name" : "Dr. Lee",
        "rank" : "Assistant Professor",
        "phone" : "555-1212"
      }
   ]
}
```

- **JSON is often used for transmitting data in across web services**
  - We will spend some time in the 4th week of the bootcamp using JSON when we explore how to interact with REST APIs

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Working with JSON

- **To transmit JSON data over a network, you must converted it to a string**
  - This is usually known as stringification
- **Then, when you want to access stringified JSON data, you must converted it back to a native JavaScript object**
  - This is usually known as parsing
- **The following JavaScript methods will help you with these tasks**
  - `JSON.stringify()`
  - `JSON.parse()`

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Stringifying JSON

- **Use `JSON.stringify()` to create a JSON string from a JavaScript object**
  - You will see this when we start calling server-side methods to retrieve data from or pass data to a REST API

Example

```
let job = {
             title : "Web Designer",
             startDate : "October 2019",
             location : "AT&T",
             minSalary : 52000,
             maxSalary : 86000
          };


let str = JSON.stringify(job);
console.log(str);

STRING CONTAINS
{
   "title" : "Web Designer",
   "startDate" : "October 2019",
   "location" : "AT&T",
   "minSalary" : 52000,
   "maxSalary" : 86000
}
```

- **You can provide a filter to `stringify()` so that it only includes the properties specified**
  - The filter is specified as a JavaScript array of quoted property names surrounded by square brackets

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Stringifying JSON *cont'd*

Example

```
let job = {
             title : "Web Designer",
             startDate : "October 2019",
             location : "AT&T",
             minSalary : 52000,
             maxSalary : 86000
          };

let filter = ["title", "location"];
let str = JSON.stringify(job, filter);

let str = JSON.stringify(job);
console.log(str);
   // only the title and location properties will be in
   // the string

STRING CONTAINS
{
   "title" : "Web Designer",
   "location" : "AT&T"
}
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Parsing JSON

- **Use `JSON.parse()` to convert a JSON string back into a JavaScript object**
  - If the string is not valid JSON, an error is generated

Example

ES6 allows multiline strings if they are surrounded by back ticks (`). This example uses this style of string for the JSON object.

```
let str = `{"title" : "Web Designer",
            "startDate" : "October 2019",
            "location" : "AT&T",
            "minSalary" : 52000,
            "maxSalary" : 86000
          }`;

let job = JSON.parse(str);
console.log(job.title);
console.log(job.location);
```

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Exercises

Add a new subfolder to `Mod04` named `JSONScripts`. The exercises in this section should be placed there.

EXERCISE

Create a script named `json_products.js`. In it, define a JavaScript object with the following properties:

```
productId
productName
unitPrice
quantity
```

Stringify the object and store it in a variable. Then display the string.

Now, write code to parse the string back into a JavaScript object and display the properties of the object to prove that it worked.

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Preparing for the Capstone and Knowledge Check

- **The following link will guide you through a thorough review of the JavaScript we have covered thus far**

- **You should review the material and practice any concepts that you feel you are weak on**

  `https://gist.github.com/erics273/acd71662cda70015b4b5443491dbfe04`

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Appendix A

Learning References

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved

Learning References

Books

Alex R. Young, et. al. *Node.js in Action*. Manning Publications, 2017.

Jon Duckett. *JavaScript and JQuery: Interactive Front-End Web Development*. Wiley, 2014.

Eric Freeman and Elisabeth Robson. *Head First JavaScript Programming: A Brain-Friendly Guide*. O'Reilly Media, 2014.

Evan Hahn. *Express in Action: Writing, building, and testing Node.js applications*. Manning Publications, 2016.

Marijn Haverbeke. *Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming*. No Starch Press, 2018.

Zak Ruvalcaba, et. al. *Murach's JavaScript and jQuery (3rd Edition)*. Mike Murach & Associates, 2017.

Online References

JavaScript:

`www.w3schools.com/js`
JavaScript tutorial and references

`developer.mozilla.org/en-US/docs/Web/JavaScript`
JavaScript tutorial and references

jQuery:

`www.w3schools.com/jquery`
jQuery tutorial and references

`jquery.com`
jQuery tutorial and references

Node.js and Express:

`www.w3schools.com/nodejs/`
Node.js tutorial and references

`expressjs.com`
Express guides and references

`developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction`
Node.js and Express tutorial and references

DRAFT

Copyright © 2019-2020 EMCA Consulting
All Rights Reserved