Learn to Code

JavaScript

Student Workbook 5

Version 5.1 Y

author: Dana L. Wyatt, Ph.D.

Table of Contents

Module 1

Arrays :
Continuing the Learning

Section 1–1

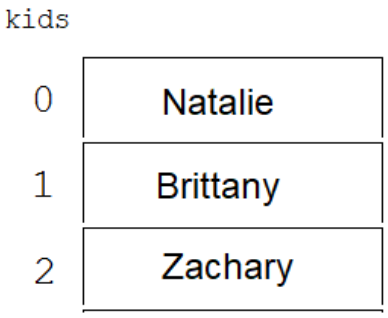
Leveraging Arrays Methods

Review of Arrays

- **We've previous learned that 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];
```

- **We've also learned that JavaScript arrays can store any type of data**

Example

```
let menu = [
      {item: "Hamburger", price: 6.95},
      {item: "Cheeseburger", price: 7.95},
      {item: "Hot dog", price: 4.95}
];
```

- **In fact, by mixing arrays and objects in creative fashions, we can represent interesting data**

Example

```
let stateInfo = [
      {
         state: "California",
         cities: ["San Diego", "San Francisco",
             "Santa Barbara", "Monterey", "Redwood City"]
      },
      {
         state: "Maryland",
         cities: ["Leonardtown", "St Leonard",
             "St Mary's City", "Annapolis", "Ocean City"]
      },
      {
         state: "Texas",
         cities: ["San Antonio", "Austin", "Pflugerville",
             "Dallas", "Ft Worth", "Paris", "Athens"]
      }
];


console.log("I live in " + stateInfo[2].cities[4] + " " +
            stateInfo[2].state);
```

Looping Though an Array of Objects

- **You've written loops to search arrays to find a single element matching a 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},
    ...
];

// find a single element matching a condition

let searchId = 4;

let matching = null;

let numItems = menu.length;
for(let i = 0; i < numItems; i++) {

   if (menu[i].id == searchId) {

      matching = menu[i];
      break;   // you've found it!  stop looking!

   }

}

if (matching != null) {
   console.log(matching.item + " costs $" + matching.price);
}
else {
   console.log("Item " + searchId + " not found!");
}
```

- **You've also written loops to search arrays to find a subset of elements that match a 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},
    ...
];

// find a subset of elements that match a condition

let searchCategory = "Drink";

let matching = [];

let numItems = menu.length;
for(let i = 0; i < numItems; i++) {

   if (menu[i].category == searchCategory) {
      matching.push(menu[i]);
   }

}

if (matching.length != 0) {
   let numMatches = matching.length;

   for(let i = 0; i < numMatches; i++) {
      console.log(
        matching[i].item + " costs $" + matching[i].price);
   }
}
else {
   console.log("No items matched category " + searchCategory);
}
```

ES 6 Search Improvements: Find First

- **ES6 introduced the `find()` method for arrays that searches the array for the *first* value that matches a specified condition**

- **To specify the condition, you pass `find()` a function object**
  - `find()` iterates over the array and calls that function for each element in the array
    - The function determines whether the array element passed "matches" the search condition
  - If the function returns `true`, `find()` stops and returns that element
    - If the function returns `false`, `find()` continues to iterate over the array
  - If the function never finds a value that matches the condition, `find()` returns `undefined`

Example

```
function isOver60(arrayValue) {
   if (arrayValue > 60) {
      return true;
   }
   else {
      return false;
   }
}

let numbers = [7, 9, 64, 60, 12, 13, 65, 62];
let firstValOver60 = numbers.find(isOver60);  // returns 64

if (firstValOver60 != undefined) {
   console.log(firstValOver60);
}
```

```
else {
   console.log("No values over 60");
}
```

- **ES6 also contains a `findIndex()` method that behaves similar to `find()` except that it returns the index of the first element that passes the test function**
  - If no values match, `-1` is returned

Example

```
function isOver60(arrayValue) {
   if (arrayValue > 60) {
      return true;
   }
   else {
      return false;
   }
}

let numbers = [7, 9, 64, 60, 12, 13, 65, 62];
let firstValOver60Position = numbers.findIndex(isOver60);
    // returns the index 2

if (firstValOver60Position != -1) {
   console.log("First value over 60 found at position " +
               firstValOver60Position);
}
else {
   console.log("No values over 60");
}
```

ES 6 Search Improvements: Find All

- **ES6 also introduced the `filter()` method for arrays that returns an array of all values that pass a test function**
  - If no values match, an empty array (`length == 0`) is returned

Example

```
function isOver60(arrayValue) {
   if (arrayValue > 60) {
      return true;
   }
   else {
      return false;
   }
}

let numbers = [7, 9, 64, 60, 12, 13, 67, 66];
let allOver60 = numbers.filter(isOver60);
    // returns an array containing [64, 67, 66]

if (allOver60.length > 0) {
   console.log(allOver60);  // displays the array
}
else {
   console.log("No values over 60");
}
```

Exercises

Create a new folder in your `LearnToCode` repo named `Workbook5`.

Create a GitHub repo named `WB5-exercises` and clone it under the `LearnToCode\Workbook5` folder.

Under it, add a subfolder named `ES6Scripts`. The exercises in this section should be placed there.

EXERCISE 1

Create a script named `es6_course_search.js`. Add a `courses` array to it that resembles the following. (You should be able to copy the text from this PDF.)

```
let courses = [
    {
        CourseId: "PROG100",
        Title: "Introduction to HTML/CSS/Git",
        Location: "Classroom 7",
        StartDate: "09/08/22",
        Fee: "100.00",
    },
    {
        CourseId: "PROG200",
        Title: "Introduction to JavaScript",
        Location: "Classroom 9",
        StartDate: "11/22/22",
        Fee: "350.00",
    },
    {
        CourseId: "PROG300",
        Title: "Introduction to Java",
        Location: "Classroom 1",
        StartDate: "01/09/23",
        Fee: "50.00",
    },
    {
        CourseId: "PROG400",
        Title: "Introduction to SQL and Databases",
        Location: "Classroom 7",
        StartDate: "03/16/23",
        Fee: "50.00",
    },
    {
```

```
        CourseId: "PROJ500",
        Title: "Introduction to Angular",
        Location: "Classroom 1",
        StartDate: "04/25/23",
        Fee: "50.00",
    }
];
```

Write code that searches the courses array using the `find()` or `filter()` functions to determine:

```
// When does the PROG200 course start?

// What is the title of the PROJ500 course?

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

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

Easier Looping? `forEach()`

- **The `forEach()` method calls a function for each element in the array**
  - That function can "process" the element

- **The `forEach()` method minimizes the need to write `for` loops**

Example

```
let kids = [
  { first : "Natalie", last : "Plyers" },
  { first: "Brittany", last: "Ray" },
  { first: "Zachary", last: "Westly" }
];

function displayKid(arrayElement) {
  console.log(arrayElement.first + " " + arrayElement.last);
}

kids.forEach(displayKid);
```

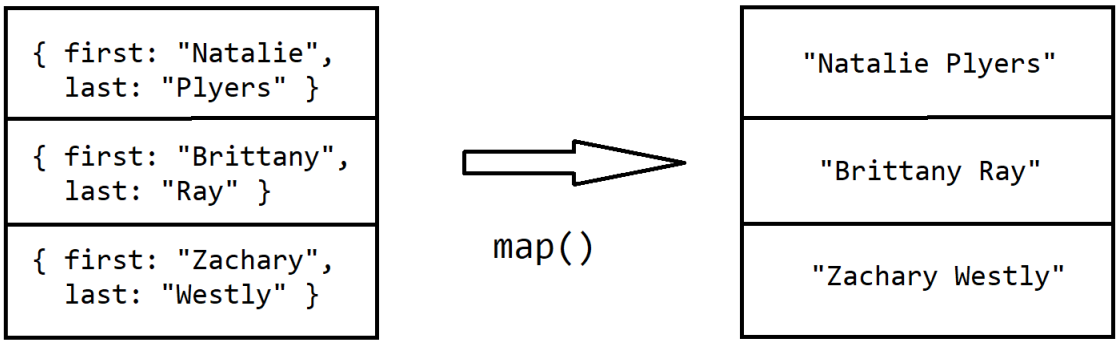
OUTPUT

```
Natalie Plyers
Brittany Ray
Zachary Westly
```

- **Some people love `forEach()`, while other people continue to write loops that iterate through arrays using `for` statements**
  - You will see a mix of these in the rest of our workbooks

Transforming Data using `map()`

- **The `map()` function is used to transform (or map) elements from one array into elements in a new array**



- **To do this, `map()` iterates over each element in the array and passes the current element to a transform function**
  - The transform function "manipulates" (or transforms) the incoming data and returns a value that is added to the new array

Example

```
let kids = [
  { first : "Natalie", last : "Plyers" },
  { first: "Brittany", last: "Ray" },
  { first: "Zachary", last: "Westly" }
];

function buildFullName(arrayElement) {
  return arrayElement.first + " " + arrayElement.last;
}

let namesList = kids.map(buildFullName);

let numElements = namesList.length;
for (let i = 0; i < numElements; i++) {
   console.log(namesList[i]);
   // output matches image above
}
```

Another Example

Example

```
let kids = [
  { first : "Natalie", last : "Plyers" },
  { first: "Brittany", last: "Ray" },
  { first: "Zachary", last: "Westly" }
];

function buildFullName(arrayElement) {
  return arrayElement.first + " " + arrayElement.last;
}

function displayName(arrayElement) {
  console.log(arrayElement);
}

let namesList = kids.map(buildFullName);

namesList.forEach(displayName);
```

reduce()

- **The `reduce()` method executes a reducer function for array elements**
  - `reduce()` returns a single value, which is the reducer's accumulated result

- **When you call `reduce()`, you pass it the reducer function and the starter value for the accumulated total**

Example

```
let numbers = [300, 150, 25];

function getSum(currentTotal, arrayValue) {
  // adds the array value to the accumulated total
  return currentTotal + arrayValue;
}

let sum = numbers.reduce(getSum, 0);
   // sum contains 475
```

- **If the array element is an object, don't forget to access the object's properties!**

Example

```
let purchasedOptions  = [
    {item: "A/C", price: 1019.00},
    {item: "Sunroof", price: 699.99},
    {item: "AM/FM Stereo w/ cassette player", price: 199.00}
];

function getTotalCost(currentTotal, arrayElement) {
  return currentTotal + arrayElement.price;
}

let sum = purchasedOptions.reduce(getTotalCost, 0);
   // sum contains 1917.99
```

- **The reducer function might be more complicated than what we just saw**
  - In the following example, it multiplies ticket price by the number of tickets before adding it to the accumulated total

Example

```
let ticketPackage  = [
    {ticketType: "Senior", price: 49.99, numTickets: 1},
    {ticketType: "Adult", price: 79.99, numTickets: 2},
    {ticketType: "Child", price: 29.99, un numTickets its: 3}
];

function getTotalCost(currentTotal, arrayElement) {
  return currentTotal +
         (arrayElement.price * arrayElement.numTickets);
}

let sum = ticketPackage.reduce(getTotalCost, 0);
   // sum contains 299.94
```

Review: Sorting an Array

- **The `sort()` method sorts an array alphabetically**
  - By default, the `sort()` method sorts the values in ascending order as strings

Example

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

kids.sort();   // Sorts the array

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

OUTPUT
Brittany
Natalie
Zachary
```

- **BUT this doesn't work well if your array contains numbers**

Example

```
let numbers = [3, 27, 400, 1, 111, 5];

numbers.sort();

// Results:  1, 111, 27, 3, 400, 5
```

- **Why did this happen? JavaScript did a "character based" sort!**
  - For more information: Google "ASCI character codes"

- **To sort arrays of numbers, you have to provide a helper method that assists the `sort()` method**
  - The helper function is called repetitively for pairs of adjacent elements
  - It should return:
    a negative number if the elements are in the right order
    0 if the elements are the same
    a positive number if the elements need to be swapped

- **You can use a named function and pass the sort function the named function**

Example

Sort numbers in an array in **ascending** order

```
function compareAscendingNumber(a, b) {
   if (a < b) return -1;          // right order
   else if (a == b) return 0;     // same values
   else return 1;                 // swap, wrong order
}

let numbers = [3, 27, 400, 1, 111, 5];

numbers.sort(compareAscendingNumber);

// Results: 1, 3, 5, 27, 111, 400
```

- **`sort()` calls the comparison function over and over with pairs of adjacent array elements**
  - The comparison function enforces the collating order by deciding which parameter is the one that goes first

- **You can also use an anonymous function and pass the function expression**

Example

```
let numbers = [3, 27, 400, 1, 111, 5];

numbers.sort(function(a, b){
   if (a < b) return -1;
   else if (a == b) return 0;
   else return 1;
});

console.log(numbers);
    // Results: [1, 3, 5, 27, 111, 400]
```

- **If you want a descending numeric `sort()`, reverse your logic!**
  - Swap if `a < b`

Example

Sort numbers in an array in **descending** order

```
let numbers = [3, 27, 400, 1, 111, 5];

numbers.sort(function(a, b){
   if (a > b) return -1;          // right order
   else if (a == b) return 0;
   else return 1;                 // swap, wrong order
});

console.log(numbers);
    // Results: [400, 111, 27, 5, 3, 1]
```

Review: Sorting Numbers and Tricks

- **You can shorten the comparison function for a numeric sort by taking advantage of math**
  - If `a - b` is a negative number, they are in the right order so don't swap the numbers

Example

Sort numbers in an array in **ascending** order

```
function compareAscendingNumber(a, b) {
   // if a is smaller, a-b is negative so don't swap!
   return a - b;
}

let numbers = [3, 27, 400, 1, 111, 5];

numbers.sort(compareAscendingNumber);

// Results: 1, 3, 5, 27, 111, 400
```

- **If you want a descending numeric `sort()`, reverse your logic!**
  - Swap if `a < b`

Example

Sort numbers in an array in **descending** order

```
let numbers = [3, 27, 400, 1, 111, 5];

numbers.sort(function(a, b){
     return b - a;
});

// Results: 400, 111, 27, 5, 3, 1
```

Review: Sorting an Array of Objects

- **This same technique can be applied to arrays of objects**
  - However, you must compare a property of the object

Example

```
let products = [
    {prodId: 2, item: "Notepads (12 pk)", price: 12.29},
    {prodId: 12, item: "Black Pens (12 pk)", price: 5.70},
    {prodId: 22, item: "Stapler", price: 12.79}
];

products.sort(function(a, b){
     if (a.item < b.item) return -1;
     else if (a.item == b.item) return 0;
     else return 1;
});

let numProducts = products.length;
for(let i = 0; i < numProducts; i++) {
   console.log(products[i].item +
               " $" + products[i].price.toFixed(2));
}
```

OUTPUT

```
Black Pens (12 pk) $5.70}
Notepads (12 pk) $12.29
Stapler $12.79
```

Exercises

EXERCISE 1

Create a script named `array_methods.js`. Add a `cart` array to it that resembles the following. (You should be able to copy the text from this PDF.)

```
let cart  = [
    {item: "Bread", price: 3.29, quantity: 2},
    {item: "Milk", price: 4.09, quantity: 1},
    {item: "T-Bone Steak", price: 12.99, quantity: 2},
    {item: "Baking Potato", price: 1.89, quantity: 6},
    {item: "Iceberg Lettuce", price: 2.06, quantity: 1},
    {item: "Ice Cream - Vanilla", price: 6.81, quantity: 1},
    {item: "Apples", price: 0.66, quantity: 6}
];
```

a. Write code that searches the courses array using the `map()` function to return only the item name and then use `forEach()` to display the list of items.

b. Write code that determines the total cost of everything in the cart using `reduce()`. In the reducer function, remember to account for the possibility of there being more than 1 unit (ex: 5 apples rather than 1)!

c. Revisit your code for (a) above. Can you sort the list before you display it?

Section 1–2

Using Arrow Functions with Array Methods

Arrow Functions

- **ES6 introduced a shorthand notation for defining functions called arrow functions**

- **They are especially useful when passing a function object to a method**

- **The arrow function, in its shortest form, might resemble**

Example

```
(a, b) => a * b

   // (a, b) is the parameter list
   //  =>    is the arrow and indicates the body follows
   //  if there is only a single expression, the value
   //  calculated is automatically returned
```

- **Imagine how you might use it with the `reduce()` method**

Example

```
let numbers = [300, 150, 25];

/*
function getSum(currentTotal, arrayValue) {
  return currentTotal + arrayValue;
}
*/

let sum = numbers.reduce((currentTotal, arrayValue) =>
    currentTotal + arrayValue
, 0);
   // sum contains 475
```

- **Imagine how you might use it with the `find()` or `filter()` methods**
  - If the condition after the arrow is true, the arrow function returns true; otherwise, it returns false

Example

```
let addOnOptions  = [
    {item: "A/C", price: 1019.00},
    {item: "Sunroof", price: 699.99},
    {item: "Mud flaps", price: 299.49},
    {item: "Heated seats", price: 1199.99},
    {item: "AM/FM Stereo w/ cassette player", price: 199.00}
];

let oneCheapOption =
   addOnOptions.find((arrayValue) => arrayValue.price < 500);

if (oneCheapOption != undefined) {
   console.log(oneCheapOption.item +
               " $" + oneCheapOption.price);
}
else {
   console.log("No options under $500");
}
```

Example

```
let people = [
   {name: "Zachary", age: 31},
   {name: "Brittany", age: 35},
   {name: "Jason", age: 36},
   {name: "Natalie", age: 37},
   {name: "Jennifer", age: 38}
];

let somePeople = people.filter(p => p.age >= 36);

   // in the code above, p is a variable that represents a
   // value in the array people; it is used in the comparison
   // as the find function looks at each value in the array

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

- **Arrow function can be used in any situation where you pass in a function object**
  - If the function requires more than one line of code, you must use curly braces (`{ }`) and use an explicit `return` statement if needed

Example

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

mybutton.onclick( () => {
   let message = "Hello World";
   alert(message);
} );
```

Exercises

EXERCISE 1

Create a script named `es6_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
];
```

Using the new methods you've learned in this module and arrow functions, write code that searches the products array to find:

```
// Which candies costs less than $4.00?

// Which candies has "M&M" its name?

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

EXERCISE 2

Create a script named `es6_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: ["Who is Fred?", "Where is Fred?",
```

```
                "What is Fred?", "Why Fred?"]
    },
    {
      memID: 203,
      name: "Bobbie Boots",
      films: ["Walking Boots", "Hiking Boots",
              "Cowboy Boots"]
    },
];
```

Using the new methods you've learned in this module and arrow functions, write code that searches the products array to find:

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

// Who has been in at least 3 films?

// Who has a name that starts with "Bob"?

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

(Bonus) EXERCISE 3

Create a script named `es6_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("8-31-2023"),
        capacity: 3
    },
    {
        color: "White",
```

```
        type: "Pickup Truck",
        registrationState: "TX",
        licenseNo: "A22-X00",
        registrationExpires: new Date("9-31-2023"),
        capacity: 6
    },
    {
        color: "Red",
        type: "Car",
        registrationState: "CA",
        licenseNo: "ABC-222",
        registrationExpires: new Date("12-10-2022"),
        capacity: 5
    },
    {
        color: "Black",
        type: "SUV",
        registrationState: "CA",
        licenseNo: "EEE-222",
        registrationExpires: new Date("12-10-2021"),
        capacity: 7
    },

    {
        color: "Red",
        type: "SUV",
        registrationState: "TX",
        licenseNo: "ZZ2-101",
        registrationExpires: new Date("5-30-2022"),
        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("3-31-2023"),
        capacity: 5
    }
];
```

Using the new methods you've learned in this module and arrow functions, write code that searches the products array to find:

```
// Which vehicles are RED?

// Which vehicles have registrations that are expired?

// Which vehicles that hold at least 6 people?

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

Section 1–3

Multidimensional Arrays

Multidimensional Arrays

- **JavaScript supports multidimensional arrays by creatively using an array of arrays**

- **A two-dimensional array is like a spreadsheet**
  - For example: think about Excel worksheet where you specify a row and a column to access a cell

- **A three-dimensional array is like an array of two-dimensional arrays**
  - For example: an Excel workbook with multiple worksheets

Example

```
let lockerAndAccessCode = [
    ["Locker 1", 135],
    ["Locker 2", 159],
    ["Locker 3", 642]
];

// access the first element
console.log(lockerAndAccessCode[0]);     // ["Locker 1", 135]

// access the first inner elemet of the second element
console.log(lockerAndAccessCode[1][0]);  // "Locker 2"
```

Example

```
let teamMembers = [
                     ["Dana", "Brenda", "Happy"],
                     ["Laura", "Patti"],
                     ["Leslye", "Randy", "Mollye", "Ranse"],
                     ["Eloise", "Robert"]
                   ];
```

```
let numTeams = teamMembers.length;
for (let i = 0; i < numTeams; i++) {
   console.log("-----------");
   console.log("Team " + (i + 1));
   console.log("-----------");

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

OUTPUT

```
"-----------"
"Team 1"
"-----------"
"Dana"
"Brenda"
"Happy"
"-----------"
"Team 2"
"-----------"
"Laura"
"Patti"
"-----------"
"Team 3"
"-----------"
"Leslye"
"Randy"
"Mollye"
"Ranse"
"-----------"
"Team 4"
"-----------"
"Eloise"
"Robert"
```

- **You can also use `forEach()` with a multidimensional array, but it is important to realize what is passed to the function that `forEach()` calls**

Example

```
let teamMembers = [
                     ["Dana", "Brenda", "Happy"],
                     ["Laura", "Patti"],
                     ["Leslye", "Randy", "Mollye", "Ranse"],
                     ["Eloise", "Robert"]
                   ];

function displayPlayer(member) {
   console.log(member);
}

let numTeams = teamMembers.length;
for (let i = 0; i < numTeams; i++) {
   console.log("-----------");
   console.log("Team " + (i + 1));
   console.log("-----------");

   teamMembers[i].forEach(displayPlayer);
}
```

Exercises

Add a subfolder under `WB5-exercises` named `MultiArrayScripts.` The exercises in this section should be placed there.

EXERCISE 1

Create a script named `multi_arrays.js`. Type the "locker and access code" just saw. Now run it.

Predict the output of the following before you run it.

```
console.log(lockerAndAccessCode[2][1]);

console.log(lockerAndAccessCode[1]);
```

Were you right?

(Challenge) EXERCISE 2

Create a script named `food_menu_array.js`. In it define a 2 dimensional array that lists items for sale for the 3 meals of the day.

```
let menu = [
              [ /* breakfast items */ ],
              [ /* lunch items */ ],
              [ /* dinner items */ ],
           ];
```

The first element in the 2-D array contains at least 3 JavaScript objects that identify breakfast items. For example:

```
let menu = [
       [
         {item: "Sausage and Egg Biscuit", price: 3.69},
         {item: "Bacon and Egg Biscuit", price: 3.49},
         {item: "Ham and Egg Biscuit", price: 3.29}
       ],
       [ /* lunch items */ ],
       [ /* dinner items */ ]
];
```

The second element in the 2-D array contains at least 4 JavaScript objects describing lunch items. The last element in the 2-D array contains at least 5 JavaScript objects describing dinner items.

Now, define a variable named `meal` that holds an integer in the range 0-2. It will represent the meal the user wants to view from the menu.

Write code to display a heading that says, "Breakfast Menu", "Lunch Menu" or "Dinner Menu" depending on the value of `meal`. Then list the items on the menu for that meal.

Test your script for each possible value of meal.

Section 1–4

`for-in`

Associative Arrays and `for-in`

- **JavaScript *objects* are sometimes called associative arrays**
  - You can find an item in an object using the property name as a subscript

Example

```
let person = {name: "Pursalane", age: 11, gender: "Female"};

console.log(person["name"]);
console.log(person["age"]);
console.log(person["gender"]);
```

  - You can iterate over the properties using the `for-in` loop
    * `for-in` returns the index or key of the next item in the array
    * We can't use a regular `for` loop because the elements are not accessible by an index

Example

```
let person = {name: "Pursalane", age: 11, gender: "Female"};

for (let key in person) {
   console.log(key + " = " + person[key]);
}

OUTPUT
name = Pursalane
age = 11
gender = Female
```

- **You may never use this feature**
  - Or you might someday find a cool use for it!

Module 2

Working in the Browser

Section 2–1

The DOM and the BOM

Executing JavaScript

- **JavaScript code is executed by a JavaScript engine**
  - The JavaScript engine might be embedded into a browser
  - The JavaScript engine might be embedded into a runtime system like Node.js
- **There are many JavaScript engines, including:**
  - V8 - a very popular engine that is used by many systems, including Chrome and Node.js
  - Chakara - used by IE9+ and Edge
    * Note: Microsoft is currently redesigning Edge to use V8
  - SpiderMonkey - used by Firefox
  - JavaScriptCode - used by Safari
- **Because browsers use different JavaScript engines, a feature available in one browser might not be supported in another**
  - You can use `https://caniuse.com` to see what browsers support a particular feature

JavaScript in the Browser

- **When you write JavaScript that runs in the browser, it is called client-side JavaScript**

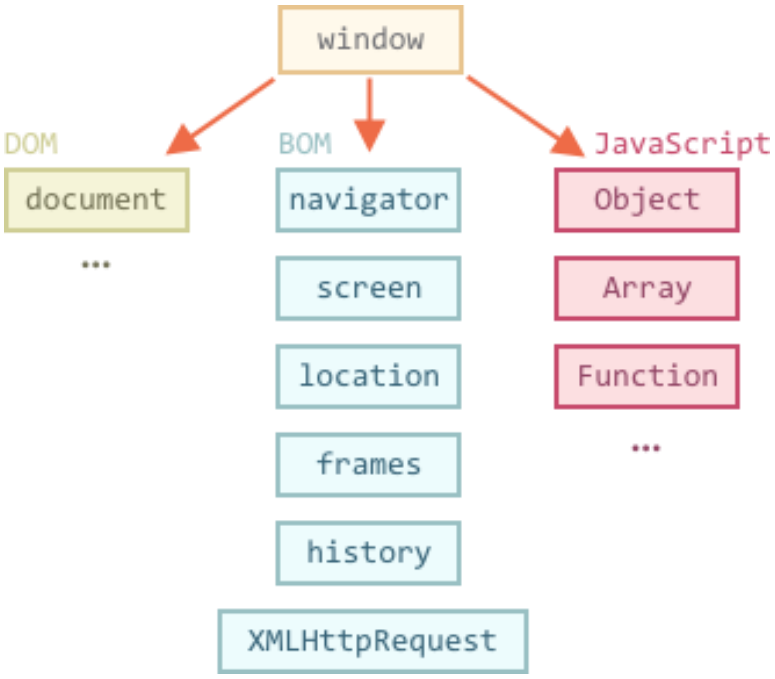
- **You've already seen that it can be used to:**
  - assign event handlers to form elements
  - interact with form fields

- **It can also be used to:**
  - interact with properties of the browser
  - control the browser by opening and closing windows
  - manipulate the page contents

- **In JavaScript, the `window` object represents the browser window and all its features**
  - There is no standard, but all major browsers support it

The DOM and the BOM

- **The `window` object gives us access to the DOM and the BOM**
  - The Document Object Model (DOM) allows JavaScript to interact with the page contents
  - The Browser Object Model (BOM) allows JavaScript to interact with the browser



Document Object Model (DOM)

- **The `document` object provides access to the page content**
  - When the HTML document is loaded into the browser, the `document` object gives you access to the root node

Example

```
const name = document.getElementById("nameField").value;

document.body.style.background = "yellow";
```

- **In the early days of the internet, there were no standards about how browsers worked**
  - This led to coding different versions of the same code just to make the program work in multiple browsers
  - Those days are "mostly" over if your users use modern browsers

- **As a client-side web programmer, you will work with the DOM continuously to perform tasks such as:**
  - Getting/setting values in form fields
  - Modifying the `innerHTML` of spans, paragraphs, and divs
  - Modifying the appearance of HTML elements by changing HTML attributes and applying CSS styles
  - Modifying the structure of the page by dynamically adding, replacing, and removing HTML elements

- **You can learn all about the DOM at: `www.w3schools.com/jsref/dom_obj_document.asp`**

Browser Object Model: The `navigator` Object

- **The `navigator` object provides information about the browser and the operating system**
  - One useful property is `navigator.userAgent` that provides information about the user's browser

Example

```
let msg = "User-agent: " + window.navigator.userAgent;
```

POSSIBLE OUTPUT FROM CHROME

```
User-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.142
Safari/537.36
```

POSSIBLE OUTPUT FROM EDGE

```
User-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.140
Safari/537.36 Edge/18.17763
```

POSSIBLE OUTPUT FROM OPERA

```
Opera/9.80 (Macintosh; Intel Mac OS X; U; en) Presto/2.2.15
Version/10.00
```

CRAWLER bot

```
Googlebot/2.1 (+http://www.google.com/bot.html)
```

- **However, placing `window.` before BOM and DOM objects is optional in JavaScript**

Example

```
let msg = "User-agent: " + navigator.userAgent;
```

Browser Object Model: The `location` Object

- **The `location` object provides access to read the current URL**

Example

```
alert(location.href);
```

- **Often, it is used to redirect the browser to a new URL**

Example

```
location.href = "https://www.traveltexas.com";
```

- **To redirect the browser to a new URL 5 seconds after the page loads, you could include the following at the bottom of the body**

Example

We will learn more about `setTimeout` later in the course.

```
setTimeout(
  function() {
     location.href = "https://www.traveltexas.com/";
  },
  5000
);
```

Section 2–2

Opening and Closing Browser Windows

Opening and Closing Browser Windows

- **You can use the `window` object to open and close browser windows**
  - `window.open()` open a new window
  - `window.close()` close the current window
- **To open a new window, call `open()` and provide a URL**
  - Most modern browsers open new tabs instead of opening separate popup windows

Example

```
window.open("https://www.w3schools.com/js/js_htmldom.asp");
```

- **Often, browsers block the window if they are opened outside of an event handler like `onclick`**

Exercises

Add a subfolder under `WB5-exercises` named `NewTabs`. These exercises should be placed there.

EXERCISE 1

Create two pages: `index.html` and `images.html`.

On the images page, place:

- An anchor tag that takes you back to `index.html`
- Two images of your choice

On the `index.html` page, add three buttons. The click event handlers of each button should call `window.open()`

- One opens the Google home page
- One opens the `www.w3schools.com/js` page
- One opens your images page

The site doesn't have to be beautiful. We are just testing how `open()` works!

CHALLENGE: You should open external sites in new tabs like we just did for Google and W3Schools. But sometimes, you want to open an internal page and replace the page that use was interacting with. Google and see if you can figure out how to use `window.open()` to open `images.html` in the same tab replacing `index.html`.

Module 3

Working with the DOM

Section 3–1

Beyond `getElementById`

Accessing an Element by `id`

- **We've already seen how you can access an element in the DOM by its `id` using `getElementsById()`**

  Example

  HTML

```
<div id="msgDiv">...</div>
```

  JavaScript

```
const msgDiv =  document.getElementById("msgDiv");
msgDiv.innerHTML = "Your order is ready.";
```

- **This is an efficient way to look up an item on a page**
  - In HTML, ids must be unique per page

Accessing Elements by Class

- **You can also access an element in the DOM by its class using `getElementsByClassName()`**

  Example

  HTML

```
<div class="msgArea">...</div>
<div class="msgArea">...</div>
```

  JavaScript

```
const msgAreas = document.getElementsByClassName("msgArea");
```

- **This method returns an `HTMLCollection` of objects with the matching class name**

- **An `HTMLCollection` is not an array, but is subscriptable**

  Example

```
const msgAreas = document.getElementsByClassName("msgArea");

let numMessageAreas = msgAreas.length
for (let i = 0; i < numMessageAreas; i++) {
   msgAreas[i].style.border = "2px solid red";
}
```

Accessing Elements by Tag Name

- **You can also access elements in the DOM by their tag name using `getElementsByTagName()`**

Example

HTML

```
<div>...</div>
<div>...</div>
```

JavaScript

```
const allDivs = document.getElementsByTagName("div");
```

- **This method returns an `HTMLCollection` of objects with the matching tag name**

Example

```
const allDivs = document.getElementsByTagName("div");

let numDivs = allDivs.length
for (let i = 0; i < numDivs; i++) {
   allDivs[i].style.border = "2px solid red";
}
```

Query Selectors

- **JavaScript also has two methods that will allow you to select elements using a CSS-style selector**
  - `querySelector()` is used when selecting a single element
    - If it matches more than 1 element, it returns only the FIRST
  - `querySelectorAll()` is used when selecting multiple elements

- **Use `querySelector()` to select by id**
  - If it returns `null`, it didn't find the element

Example

```
const nameField = document.querySelector("#nameField");
let name = nameField.value;
```

- **Use `querySelectorAll()` when you want all of the elements matching the specified selector**
  - It returns an `HTMLCollection` of objects that matched the specified selector

- **Selecting by class name returns 0 or more elements**

Example

```
const msgAreas = document.querySelectorAll(".msgArea");

let numMessageAreas = msgAreas.length;
for (let i = 0; i < numMessageAreas; i++) {
   msgAreas[i].style.border = "2px solid red";
}
```

- **Selecting by tag name returns 0 or more elements**

Example

```
const allDivs = document.querySelectorAll("div");

let numDivs = allDivs.length;
for (let i = 0; i < numDivs; i++) {
   allDivs[i].style.border = "2px solid red";
}
```

- **Using the query selector methods is extremely powerful**
  - You can access any element or group of elements in the DOM the same way you would in a CSS file

- **For a complete list of CSS selectors, see:**
  `https://www.w3schools.com/cssref/css_selectors.asp`

Using the `forEach()` Method

- **We saw earlier that you could use the `forEach()` method to iterate through elements in an array**
  - It loops through the array and calls a specified function for each element in that array

Example

```
function display(item) {
   console.log(item);
}

let colors = ["Red", "White", "Blue"]
colors.forEach(display);
```

OUTPUT

```
Red
White
Blue
```

Example

```
let colors = ["Red", "White", "Blue"]

colors.forEach(function (item) {
   console.log(item);
});
```

Example

```
let colors = ["Red", "White", "Blue"]

colors.forEach((item) => {
   console.log(item);
});
```

- **If you have an `HTMLCollection`, you must call `Array.from()` on the collection before using `forEach`**
  - This is because an `HTMLCollection` is NOT an array even though it is subscriptable

Example

```
const msgDivs = document.querySelectorAll(".msgDiv");

Array.from(msgDivs).forEach(function (element) {
   element.style.border = "2px solid red";
});
```

Example

```
const msgDivs = document.querySelectorAll(".msgDiv");

Array.from(msgDivs).forEach((element) => {
   element.style.border = "2px solid red";
});
```

Exercises

EXERCISE 1

What does each return?

```
let objSet1 = document.querySelectorAll("p");


let objSet2 = document.querySelectorAll(".attn");


let objSet3 = document.querySelectorAll("p.attn");


let objSet4 = document.querySelectorAll("img[alt]");


let objSet4 = document.querySelectorAll("div > p");


let objSet5 = document.querySelectorAll("div, span");
```

Now, create a CSS class rule in a `.css` file named `imageGroup`. The class simply adds a little padding to a `div`. Include it in your HTML page and then assign it to each of the `divs`.

```
<div class="imageGroup">
   <img src="images/something.png" />
   <p>describe image above</p>
</div>
```

Back in the `window.onload` event, now use `getElementsByClassName()` to find elements with the `imageGroup` class. Then programmatically add a thick red border to the elements.

Test your page.

Finally, use `querySelectorAll()` to find all images that do NOT have an alt text and then assign them the text "graffiti image". The `not()` selector is useful

in cases like this. It accepts a selector and finds everything that DOESN'T match. For example:

```
not(p)           Anything that is NOT a <p>
a:not([class])   All <a> that do NOT have a class attribute
div:not([id])    Any <div> that does not have an id
```

Test your page.

Section 3–2

Working with Elements

Working with `value` and `innerHTML`

- **As we've seen already, form fields such as `<input>` and `<select>` elements have a `value` property**
  - You can get or set the field's value

Example

HTML

```
<input type="text" id="nameField" name="name" />
<input type="number" id="ageField" name="age" />
<select id="genderField" name="gender">
   <option value="">Select One…</option>
   <option value="F">Female</option>
   <option value="M">Male</option>
</select>
```

JavaScript

```
const nameField = document.getElementById("nameField");
const ageField = document.getElementById("ageField");
const genderField = document.getElementById("genderField");

// getting values
let name = nameField.value;
let age = Number(ageField.value);
let gender = genderField.value;  // "", "F", or "M"

// setting values
nameField.value = "";
ageField.value = "";
genderField.value = "";  // reset to "Select One…"
```

- **And with elements such as `<div>`, `<p>`, and `<span>` you use `innerHTML`**

Example

```
const divMsg = document.getElementById("divMsg");
divMsg.innerHTML = "An error occurred!";
```

Accessing Properties of Elements

- **You can read or write the value of any element's HTML properties**

Example

```
// setting up an image
const img = document.getElementById("dogImg");
img.src = "images/corgi.jpg";
img.alt = "This is a photo of a really cute corgi!";
```

- **Easy, right?**

- **However, be careful because sometimes the JavaScript property name is different from the HTML attribute**
  - This happens when the HTML attribute name is a reserved word in JavaScript
  - For example: In JavaScript, we use `className` to reference the HTML `class` attribute

Example

```
const img = document.getElementById("dogImg");
img.className = "borderedImg";
```

CSS Properties

- **Every property used in CSS can be set using JavaScript**
  - Usually, the name of the JavaScript property is the same as the CSS property
    - Examples: `color` or `left`
- **However, sometimes the JavaScript name is different than what you used in CSS**
  - If the CSS property name has a hyphen or if it matches a JavaScript reserved word, then JavaScript uses a different name
- **If the CSS property has a hyphen, the JavaScript name omits the hyphen and the letter immediately following is capitalized**

CSS Property	JavaScript Property
`background-color`	`backgroundColor`
`font-size`	`fontSize`

- **If the CSS property is a reserved word, the JavaScript name is prefaced with `css` and the first letter is capitalized**
  - The CSS `float` property is named `cssFloat` in JavaScript
- **You can find good lists at:**

`https://www.w3schools.com/jsref/dom_obj_style.asp`

`http://www.sitestepper.be/en/css-properties-to-javascript-properties-reference-list.htm`

Scripting Inline Styles

- **You can use the `style` property of an element to change its inline style**
  - Remember: inline styles override any other styles

Example

```
const msgDiv = document.getElementById("msgDiv");
msgDiv.innerHTML = "Your request did not go through!";
msgDiv.style.border = "1px solid red";
msgDiv.style.color = "red";
```

- **You can also use the `cssText` property to access the inline style as a string**
  - Separate attributes and values with a colon

Example

This allows you to set several properties in one line of code.

```
const msgDiv = document.getElementById("msgDiv");
msgDiv.innerHTML = "Your request did not go through!";
msgDiv.style.cssText = "border: 1px solid red; color: red;
```

- ***NOTE: You will generally try to avoid using inline styles in most web pages!!***

Working with CSS Classes

- **You can set the class name of an element**
  - But this might be dangerous! You can delete any other classes assigned to the element

Example

```
const img = document.getElementById("img");
img.className = "borderedImg";
```

- **It is better to add or remove a class from the class list that is already assigned to the element using `classList`**

Example

```
const myDiv = document.getElementById("myDiv");
myDiv.classList.add("myStyle");
```

Example

Similar logic applies to removing a class.

```
const myDiv = document.getElementById("myDiv");
myDiv.classList.remove("myStyle");
```

Exercises

EXERCISE 1

Create a folder under `WB5-exercises` named `ImagesDemo`. Create a simple web page with three copies of the `div` shown below. ***Do NOT include alt text for the images.***

```
<div>
   <img src="..." />                    ← you select an image
   <p>describe image above</p>
</div>
```

In the `window.onload` event, use `getElementsByTagName()` to find all paragraphs and programmatically add a thin black border around them using the `style` property.

Test your page.

Now, create a CSS file and within it add CSS class rule for your images that will give them an oval appearance. Name it `roundedImg`.

Learn more about image styles at:
`https://www.w3schools.com/css/css3_images.asp`

Now, back in your `onload` event handler, apply that `roundedImg` class to your image and set its `src` and `alt` attributes.

Test your page.

Section 3–3

Modifying Page Content

Adding, Removing and Replacing Nodes

- **Very often, you will find you need to dynamically create HTML!**
  - There are several ways you can add/remove/replace nodes in the HTML document
- **If you want to work with a *child element* of a *parent container*, you can use several methods, including:**
  - `appendChild()` adds the node to the end of the list of children of a specified parent node
  - `removeChild()` method removes a child node from a specified parent node
  - `replaceChild()` method replaces a child node in a specified parent node with a different node
- **If you want to work with a *sibling element* of another *element*, you can use:**
  - `insertBefore()` inserts a node before the referenced node as a child of a specified parent node
  - Surprisingly, there is no `insertAfter()` method, but with a little work, you can code your own helper
- **You should research these methods in more detail at W3Schools before attempting any exercises!**

Appending a Child Node

- **To append a new child node to a parent node, you must do two things:**
  - Create the node
    - It will be either an ***HTML element*** or a ***text node***
  - Add the node to the document by calling `appendChild` on the parent node

- **To create an HTML element, use `createElement()`**

Example

HTML

```
<div id="locationDiv"> </div>
```

JavaScript

```
let img = document.createElement("img");
img.src = "images/Hartford.jpg";
img.alt = "A map of downtown Hartford";

const locationDiv = document.getElementById("locationDiv");
locationDiv.appendChild(img);
```

- **To create a text node, use `createTextNode()`**

Example

HTML

```
<div id="messageDiv"> </div>
```

JavaScript

```
let orderNum = 10123;
let message = "Your order number is " + orderNum;
```

```
let messageNode = document.createTextNode(message);

const msgDiv = document.getElementById("msgDiv");
msgDiv.appendChild(messageNode);
```

Removing a Child Node

- **You can also remove a child node from the document**
  - Call the `removeChild()` method on the parent of the node you want to remove

•Example

HTML

```
<div id="imagesDiv">
   <img src="images/store.jpg" alt="The store" />
   <img src="images/girl.jpg" alt="Girl's jacket">
   <img src="images/boy.jpg" alt="Boy's jacket">
   <img id="saleImg"
        src="images/sale.jpg" alt="Girl in a jacket">
</div>
```

JavaScript

```
// find the element to remove
let saleImg = document.getElementById("saleImg");

// find the parent of the element to remove
let imagesDiv = document.getElementById("imagesDiv");
imagesDiv.removeChild(saleImg);
```

Replacing a Child Node

- **You can also replace a node in the document with another node**
  - Call the `replaceChild()` method on the parent of the node you want to replace

Example

HTML

```
<div id="imagesDiv">
   <img src="images/store.jpg" alt="The store" />
   <img src="images/girl.jpg" alt="Girl's jacket">
   <img src="images/boy.jpg" alt="Boy's jacket">
   <img id="saleImg"
        src="images/sale.jpg" alt="Girl in a jacket">
</div>
```

JavaScript

```
// create the new element
let closeoutImg = document.createElement("img");
closeoutImg.src = "images/lastMinuteSale.jpg";
closeoutImg.alt = "55% off close-out items";

// find the element to be replaced
let saleImg = document.getElementById("saleImg");

// find the parent and then replace the element
let imagesDiv = document.getElementById("imagesDiv");
imagesDiv.replaceChild(closeoutImg, saleImg);
```

Exercises

EXERCISE 1

Create a folder under `WB5-exercises` named `DynamicImages`.

In this exercise, you will build a web site that allows a user to pick an image description from the dropdown list and then you will programmatically create an `<img>` element and add it to the images already shown.

They can click a "clear" button to remove all the images. At any given time, the user could be viewing from 0 to all the images.

Start by finding 5 images and size them to be the same size ~ approximately 300 H x 200 W. Places your sized images in a folder named `images`.

Now, design your `index.html` page so that it contains the following:

- a dropdown list with labeling that instructs the user to select an image
- two buttons the user can click: one adds the selected image to a div and one clears all images (note: you might see several images depending on the user's actions)
- a div that will hold the images

Place your image descriptions and file names in an array named `imageFiles` in your script at the global level. For example:

```
let imageFiles = [
  {name: "images/dog.jpg", description: "A cute dog"},
  {name: "images/cat.jpg", description: "A cute car"},
  ...
];
```

In the `window.onload` event, loop through `imageFiles` and create `option` elements for each description and add them to the dropdown list. Also, attach event handlers to each of the button's click events.

In the click event handler for showing an image:

- determine the selected item (description) in the dropdown
- find the matching file name for the image by searching `imageFiles`

- dynamically create an `<img>` element and set the `src` and `alt` properties appropriately
- use `appendChild()` to add the image to the images div

Test your page.

In the click event handler for the clear button, write code to delete the images in the images div. Deletion can be as easy as setting the div's `innerHTML` property to a empty string.

Test your page again.

Working with Tables

- **HTML tables have some methods that make working with them a little easier**
  - We will code this together in a minute

- **In the simplest scenario:**
  - You call `insertRow()` on the table to create a new row
    - The number in parenthesis specifies where the row will be added; if you specify -1, it will be added at the end of the table
  - You call `insertCell()` on the table row to create a new cell
    - The number in parenthesis specifies where the cell will be added on the table row
  - You use the `innerHtml` property on the cell to configure its contents

Example

HTML

```
<body>
     <h3>Outstanding Winning Lottery Tickets</h3>
     <table id="winningTicketsTbl">
          <tr>
               <th>Ticket Number</th>
               <th>Prize Amount</th>
               <th>Expires</th>
          </tr>
     </table>

     <script src="scripts/index.js"></script>
 </body>
```

JavaScript

```
let winningTickets = [
   {tixNum: "1001001", expires: "2022-09-05", prize: 100000},
   {tixNum: "1298711", expires: "2022-10-10" , prize: 250000},
    // others not shown
];

window.onload = function () {
  loadWinningTicketsTable();
};

function loadWinningTicketsTable() {
  // Find the table
  let table = document.getElementById("winningTicketsTbl");

  // loop through the array
  let numWinningTickets = winningTickets.length;
  for(let i=0; i < numWinningTickets; i++) {

    // Create an empty <tr> element and add it to the last
    // position of the table
    let row = table.insertRow(-1);

    // Create new cells (<td> elements) and add text
    let cell1 = row.insertCell(0);
    cell1.innerHTML = winningTickets[i].tixNum;

    let cell2 = row.insertCell(1);
    cell2.innerHTML = "$" + winningTickets[i].prize.toFixed(2);

    let cell3 = row.insertCell(2);
    cell3.innerHTML = winningTickets[i].expires;
  }
}
```

- **The output looks as expected**

Outstanding Winning Lottery Tickets

Ticket Number	Prize Amount	Expires
1001001	$100000.00	2022-09-05
1298711	$250000.00	2022-10-10
1309182	$500000.00	2022-12-30
1456171	$1000000.00	2023-01-20
3332871	$1000000.00	2022-05-20
4651529	$100000.00	2022-12-15

Leveraging Helper Functions

- **We can improve the code a bit by writing a helper function**

Example

```
let winningTickets = [
   {tixNum: "1001001", expires: "2022-09-05", prize: 100000},
   {tixNum: "1298711", expires: "2022-10-10" , prize: 250000},
    // others not shown
];

window.onload = function () {
  loadWinningTicketsTable();
};

function loadWinningTicketsTable() {
  // Find the table
  let table = document.getElementById("winningTicketsTbl");

  // loop through the array
  let numWinningTickets = winningTickets.length;
  for(let i=0; i < numWinningTickets; i++) {
     buildTicketRow(table, winningTickets[i]);
  }
}

function buildTicketRow(table, theTicket) {
    // Create an empty <tr> element and add it to the last
    // position of the table
    let row = table.insertRow(-1);

    // Create new cells (<td> elements) and add text
    let cell1 = row.insertCell(0);
    cell1.innerHTML = theTicket.tixNum;

    let cell2 = row.insertCell(1);
    cell2.innerHTML = "$" + theTicket.prize.toFixed(2);

    let cell3 = row.insertCell(2);
    cell3.innerHTML = theTicket.expires;
}
```

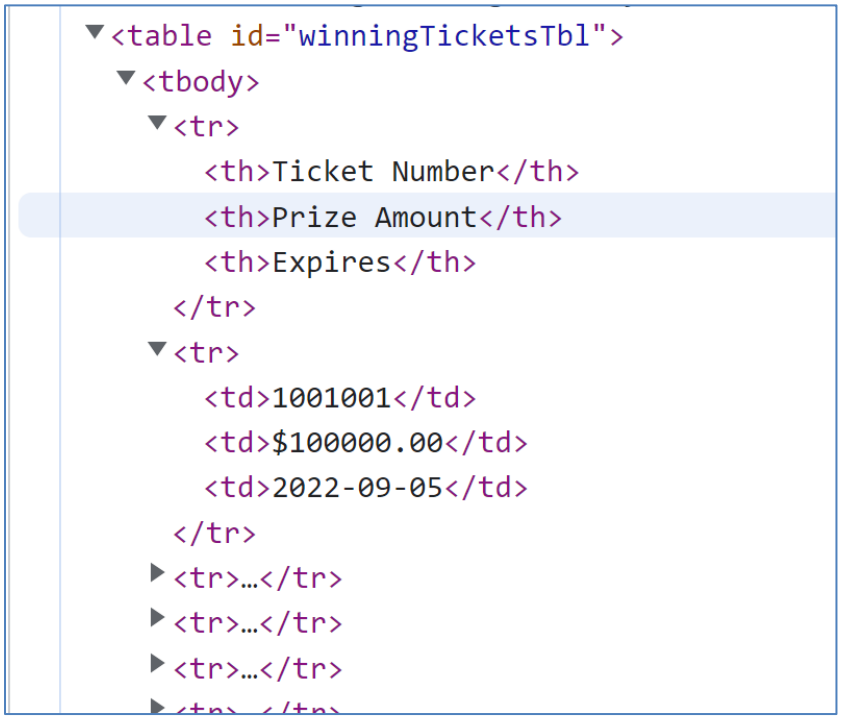
Table Structure

- **Our code is better and the output looks the same**

Outstanding Winning Lottery Tickets

Ticket Number	Prize Amount	Expires
1001001	$100000.00	2022-09-05
1298711	$250000.00	2022-10-10
1309182	$500000.00	2022-12-30
1456171	$1000000.00	2023-01-20
3332871	$1000000.00	2022-05-20
4651529	$100000.00	2022-12-15

- **But the internal HTML structure might be a little different than what you expected**
  - The browser added a `tbody` tag around all the rows, including the header!



- **If you didn't realize this, it might be hard to get your desired styling to work on the table**

- **In the code-along demo that follows, we will fix this problem by:**
  - designing the table with `thead` and `tbody` elements
  - modifying our code to add rows to the `tbody` element instead of the table element

Code-Along Demo

We will start with a code-along demo and then repeat a similar lab as an individual exercise.

In this project, we will actually implement the example we just saw.

Step 1: Let's put this project in its own repo! Create a GitHub repo named `LotteryTickets`. Clone it into your `WebProject` folder.

Step 2: Design the web site structure.

Create folders for your scripts, CSS files and images!

Step 3: Design the `index page`.

What are the standard HTML elements you need to add to the page regardless of what it does?

Create the basic page structure and include Bootstrap.

Step 4: Design the table and placeholder for your rows

In the page body, add ***only*** the following HTML elements:

```
<h3>Outstanding Winning Lottery Tickets</h3>
<table id="winningTicketsTbl">
   <thead>
      <tr>
         <th>Ticket Number</th>
         <th>Prize Amount</th>
         <th>Expires</th>
      </tr>
   </thead>
   <tbody id="winningTicketsTblBody">
   </tbody>
</table>
```

Step 5: Set up the data and the load event code

Copy the following array from the PDF to the top of your script file just below the "use strict".

```
let winningTickets = [
    {tixNum: "1001001", expires: "2022-09-05", prize: 100000},
    {tixNum: "1298711", expires: "2022-10-10" , prize: 250000},
    {tixNum: "1309182", expires: "2022-12-30" , prize: 500000},
    {tixNum: "1456171", expires: "2023-01-20", prize: 1000000},
    {tixNum: "3332871", expires: "2022-05-20", prize: 1000000},
    {tixNum: "4651529", expires: "2022-12-15", prize: 100000},
    {tixNum: "5019181", expires: "2023-01-31", prize: 250000},
    {tixNum: "5168261", expires: "2023-03-01", prize: 1000000},
    {tixNum: "6761529", expires: "2022-12-15", prize: 250000},
    {tixNum: "7778172", expires: "2023-01-15", prize: 5000000},
    {tixNum: "8751426", expires: "2020-09-15", prize: 100000}
];
```

Now, create a `window.onload` event handler. In it, call a soon-to-be-written function named `loadWinningTicketsTable()`.

*Step 6: Loading the table with the data from **`winningTickets`***

PART 1

Write the `loadWinningTicketsTable()` function.

Within it, use `document.getElementById()` to find `winningTicketsTblBody` IMPORTANT: ***find the `tbody` element instead of the table!***

Then, create a `for` loop to iterate through `winningTickets`. In that loop, call a helper (named `buildTicketRow`) that builds the row. Pass the function:

- the `tbody` element
- the current winning ticket, ex: `winningTickets[i]`

PART 2

Write the helper function named `buildTicketRow()` that receives two parameters: the `tbody` element and a single winning ticket.

```
function buildTicketRow(tbody, theTicket) {

}
```

Inside the function:

- Create a `tr` object by calling `insertRow()` on your `tbody` element (the parameter passed to the function)
- Create a `td` object and place the ticket number of the winning ticket object *again, the parameter passed to the function).
- Don't forget to add the `td` to your `tr` object.
- Repeat by creating `td`s for the prize and expires data as well

If you are confused, refer to the code example shown just before this demo.

Test your page. If you inspect your page now, your table will be well formed and easier to style.

Exercises

Let's put this project in its own repo! Create a GitHub repo named `Adventures`. Clone it into your `WebProject` folder.

EXERCISE 1

Let's repeat last week's workshop but with a different UI. In this site, you will load a dropdown list with the categories found in your tourist bureau. Don't forget to add a "Select one…" option programmatically at the top of the dropdown before adding the items in the following array.

```
let categories = [
   "Adventures",
   "Arts & Crafts",
   "Local Sports",
   "Museums",
   "Wine Tastings",
   "Other"
];
```

When the user selects a category, find the matching activities. Use the array from last week, but remember that it resembles:

```
let activities = [
   {
        category: "Adventures",
        id: "A101",
        name: "Valley Hot Air Balloons",
        description: "Enjoy a lovely hot air balloon ride over
the valley at sunrise.  Call 800-555-1212 to reserve a
date/time after you complete your e-ticket purchase.",
        location: "121 S. Main Street",
        price: 265.00
   },
   /* others not shown
];
```

When the user selects an activity category, show the activities that match in an HTML table.

There will be several issues you have to take care of, such as:

- formatting data in the table in a readable way
- clearing the displayed table when a new category is selected
- showing reasonable output when the user selects "Select one…" from the category list

BONUS: Add 'Local sports' to the categories array. Then, show reasonable output when the user selects "Local sports" (why is this different? Because there are no matching local sports!)

EXERCISE 2

Let's put this project in its own repo! Create a GitHub repo named `PeopleListing`. Clone it into your `WebProject` folder.

In this exercise, create a project that displays people in a dynamic table when the page loads. The array will be distributed by your instructor, but resembles the following:

```
let people = [
  {
    id: 1,
    firstName: "Ezra",
    lastName: "Aiden",
    email: "e.aiden@basshall.com",
    ipAddress: "18.6.24.104"
  },
  {
    id: 2,
    firstName: "Ian",
    lastName: "Auston",
    email: "ian.auston@goldmansachs.com",
    ipAddress: "17.16.4.105"
  },
  ...
];
```

You application should display the people in a HTML `<table>` when the page loads. Once it works, use the Bootstrap classes `table` and `table-striped` on the table to improve the appearance.

(Optional) EXERCISE 3

Let's put this project in its own repo! Create a GitHub repo named `EmployeeAndProjects`. Clone it into your `WebProject` folder.

Create a project that displays employee data. The array will be distributed by your instructor, but resembles the following:

```
let employees = [
  {
    id: "55007",
    name: "Karina Chambers",
    jobTitle: "Tech Lead",
    yearsAtCompany: 14,
    email: "karinachambers@ziore.com",
    wfhAdress: "640 Boynton Place, Faxon, Kentucky, 42071",
    skillSet: "Velit commodo voluptate id est. Fugiat magna enim quis
exercitation duis fugiat non nisi consequat.",
    projectsAssignedTo: [
      {
        projectId: 112,
        name: "Cupidatat aute"
      }
    ]
  },
  {
    id: "23810",
    name: "Kasey Bowers",
    jobTitle: "Senior Programmer",
    yearsAtCompany: 19,
    email: "kaseybowers@ziore.com",
    wfhAdress: "969 Clarendon Road, Marshall, Vermont, 47859",
    skillSet: "Est et voluptate incididunt deserunt culpa
excepteur.",
    projectsAssignedTo: [
      {
        projectId: 124,
        name: "Amet do deserunttate aliqua"
      }
    ]
  },
  ...
]
```

Place each employee's name into a dropdown list when the page loads. BONUS: Have the names sorted!

When an employee is selected from the dropdown, search the array for that employee and display their data in a table. This table will be different from the other ones you've done because ***each row will show only one field***. For example:

ID	72054
Name	Kendra Perry
Job Title	Web Designer
etc	

Don't show the projects they are working on. Instead, show a row that lists "How Many" projects they are assigned to.

Now, find a way to display the projects that the employee works on.

(Optional) EXERCISE 4

Let's put this project in its own repo! Create a GitHub repo named `FancyPets`. Clone it into your `WebProject` folder.

In this exercise, create a project that display all the pets listed in the array. The trick this time is that each pet should be in a Bootstrap card! The array will be distributed by your instructor, but resembles the following:

```
let pets = [
  {
    name: "Rubby",
    type: "Dog",
    breed: "Corgi",
    bestTrick: "Roll over",
    image: "images/rubby.com"
  },
  {
    name: "KitKit",
    type: "Cat",
    breed: "Mixed",
    bestTrick: "Commanding his owner to feed him",
    image: "images/kitkit.com"
  },
  ...
];
```