# Learn to Code

## JavaScript

### Workbook 4

Version 5.1 Y

author: Dana L. Wyatt, Ph.D.

# Table of Contents

# Module 1

# Working with JavaScript Objects

# Section 1–1

# JavaScript Objects

# 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 it**
  - For example, an Employee might have the following properties:
    - `employeeId`
    - `name`
    - `jobTitle`
    - `payRate`

# JavaScript Objects

- **JavaScript allows you to declare objects 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: "Theater Teacher",
              payRate: 38.46
          };
```

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

**Example**

```
let emp = { employeeId:"1", name:"Ezra",
            jobTitle:"Theater Teacher", payRate:38.46 };
```

# Accessing Object Properties

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

  **`objectName.propertyName`**

  **`objectName["propertyName"]`**

**Example**

```
let emp1 = {
               employeeId: "1",
               name: "Ezra",
               jobTitle: "Theater Teacher",
               payRate: 38.46
           };

let emp2 = {
               employeeId: "2",
               name: "Elisha",
               jobTitle: "Game Programmer",
               payRate: 43.27
           };


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**

# 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: "Theater Teacher",
              payRate: 38.46
           };

let emp2 = {
              employeeId: "2",
              name: "Elisha",
              jobTitle: "Game Programmer",
              payRate: 43.27
           };

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

- **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", 38.46, 49);

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


let emp2PayStub =
   createPayStub("2", "Elisha", 43.27, 42);

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

# Exercises

Create a new folder in your `LearnToCode` repo named `Workbook4`.

Create a GitHub repo named `WB4-exercises` and clone it under the `LearnToCode\Workbook4` folder.

Under it, 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
```

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);
```

DON'T FORGET TO commit and push your repo.

# Module 2

# Working with Loops and Arrays

# Section 2–1

# Loop Basics

# 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

# `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 say 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)
```

# `do/while` loop

- **The `do/while` is like 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)
```

# `for` loop

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

- **It has three parts separated by semicolons:**
  - code that executes 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

- **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)
```

# 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!

# Exercises

Create a new folder `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!

DON'T FORGET TO commit and push your repo.

# Section 2–2

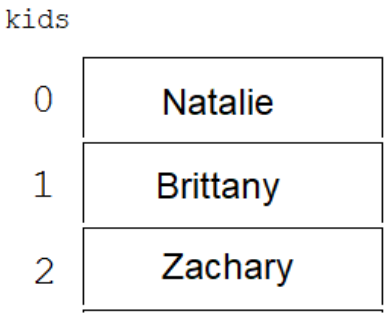
# Arrays

# 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];
```

- **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

**Example**

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

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

# 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, 9, 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];
```

# 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
```

# 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));
```

# Exercises

Create a new subfolder 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 through 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.

EXERCISE 4

Write a script named `student_average.js` that declares an array of student objects. The array contains an array of students. Each student has an array of exercise scores. For example,

```
let students = [
        {name: "Zephaniah", scores: [100, 96, 99, 92]},
        {name: "Pursalane", scores: [92, 89, 96, 100, 94]},
        {name: "Siddalee", scores: [86, 72, 92]},
        {name: "Ian", scores: [98, 84, 89, 100, 100, 76]},
        {name: "Elisha", scores: [89, 100]},
        {name: "Ezra", scores: [100, 99, 100, 87]}
];
```

This exercise will need two loops. The outer one will loop from one student to the next.

The inner loop will have to loop over a particular student's scores, add them up, and then divide by the number of scores.

DON'T FORGET TO commit and push your repo.

# 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"]
```

# Searching an Array

- **There are two functions that make searching an array 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);
```

- **`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**

```
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
```

# Finding 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));
}
```

# Exercises

## EXERCISE 1

Create a script named `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 to find:

```
// 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"?
```

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

Write code that searches the array to find:

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

// Who has have 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 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("8-31-2023"),
    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("12-30-2022"),
    capacity: 5
},
{
    color: "White",
    type: "Pickup Truck",
    registrationState: "TX",
    licenseNo: "CAC-7YT",
    registrationExpires: new Date("1-31-2023"),
    capacity: 5
},
{
    color: "White",
    type: "Pickup Truck",
    registrationState: "CA",
```

```
        licenseNo: "123-ABC",
        registrationExpires: new Date("3-31-2023"),
        capacity: 5
    }
];
```

Write code that searches the 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"?
```

DON'T FORGET TO commit and push your repo.

# Section 2–3

# Sorting Arrays

# 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 function as an argument to the sort method**

**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]
```

# 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
```

# Sorting an Array of Objects

- **This same technique can be applied to arrays of objects**
  - However, you must compare a property or properties 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
```

# A Case-Insensitive Sort

- **The collating sequence of letters is such that:**

```
'A'   <   'Z'   <   'a'   <   'z'
```

- **This means that:**

```
'Adam'   <   'Zachary'   <   'anderson'   <   'zeb'
```

- **If you want to sort these names so that they will come out in a case insensitive way (shown below), you should convert the text to the same case (upper or lower) and then do the comparison**

**Example**

```
let names = ["zeph", "anderson", "Zachary", "Adam"];

names.sort(function (a,b) {
   let aUpper = a.toUpperCase();
   let bUpper = b.toUpperCase();

   if (aUpper < bUpper) return -1;
   else if (aUpper == bUpper) return 0;
   else return 1;
});   // Sorts the array in a case-insensitive way

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

OUTPUT

```
Adam
anderson
Zachary
zeph
```

# Exercises

Create a folder named `Sorting`. These exercises will be located in that folder.

EXERCISE 1

Add a script named `courses_sorting.js`.

Take the `courses` array from an earlier lab and sort it by course title. Then display the list.

EXERCISE 2

Add a script named `products_sorting.js`.

Create a script that takes the `products` array from an earlier lab and sorts it by product name. Then display the list.

Use `console.log()` to draw a long dashed line (-------)

Now sort it by descending price. Then display the list again.

(Challenge) EXERCISE 3

Find the script that you recently wrote that finds the average score of a set of exam scores using a function named `getAverage()`. Add an additional function to the script called `getMedian()` that finds the median value in the array. The median value in the array is the "middle" element after sorting the array. However, there is a twist.

If the array has an odd number of elements, it will be the middle element:

```
ex:  [ 80, 83, 86, 92, 100]          median = 86
```

But if the array has an even number of elements, it will be the average of the middle two elements:

```
ex:  [ 80, 83, 86, 87, 92, 100]      median = 86.5
```

Before you begin, come up with an algorithm for finding the median. If you have trouble, ask your instructor.

Make sure to call the function twice: once with an odd number of scores and once with an even number of scores. Display your median along with your average.

Now, add two more display statements to display the highest and lowest score the student made. Since your array is now sorted, this should be easy! Re-test your script.

DON'T FORGET TO commit and push your repo.

# Section 2–4

# `for-of`

# `for...of` Loop

- **ES6 released the `for`…`of` statement and it is specifically designed to loop through arrays**
  - It can also loop through other collections like DOM nodes

**Syntax**

During each iteration of the loop, *`someVariable`* references the next element in *`someArray`*

```
for(someVariable of someArray) {
   // use someVariable
}
```

**Example**

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

for (let value of kids) {
   console.log(value);
}
```

OUTPUT

```
Natalie
Brittany
Zachary
```

- **Because this is a "newer" loop construct, it hasn't been fully implemented in all browsers**

`https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...of`

# Module 3

# Working with Forms
# - Part 3

# Section 3–1

# Working with Lists

# 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>
```

# 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++) {

      // create the option element
      let theOption = document.createElement("option");

      // set the text and value of the option you created
      theOption.textContent = states[i];
      theOption.value = abbrev[i];

      // append the option as a child of (inside) the
      // select element
      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**

**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++) {

      // create the option element and set the text and
      // value at the same time
      let theOption = new Option(states[i], abbrev[i]);

      // append the option as a child of (inside) the
      // select element
      statesList.appendChild(theOption);
   }

   // other stuff
   ...

};
```

# Better Design

- **A better design would be to encapsulate the dropdown loading into a function and then call it from your window's `onload` event handler**

**Example**

```
window.onload = function() {

   // load the dropdown list
   initStatesDropdown();

   // other stuff
   ...

};

function initStatesDropdown() {

   // 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++) {

      // create the option element and set the text and
      // value at the same time
      let theOption = new Option(states[i], abbrev[i]);

      // append the option as a child of (inside) the
      // select element
      statesList.appendChild(theOption);
   }
}
```

# Exercises

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

EXERCISE 1

Create an `index.html` file.

This web page will:

1. allow the user to select a football team from a dropdown list
2. allow the user to click a button after selecting the team
3. show information about the selected team in a paragraph below the button

Take a few minutes to write the HTML code for the page and commit that work.

In this exercise, we will use JavaScript to load the dropdown programmatically. **We will save other activities for the next lab.**

Add a script file for this page. Within it, 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, call a function to initialize the football `select` element.

In the initialize function, 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. Do you see the names of the teams? Commit your work.

# Determining the Option Selected

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

- **Use the `value` property of the selected option**
  - It returns `null` if nothing is selected in the dropdown

**Example**

We are going back to the example that had a dropdown of state names with the values being state abbreviations. Ex: Alabama / AL, Alaska / AK, Arizona / AZ, etc

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

let selectedValue = statesList.value;

if (selectedValue == null) {
   alert("No state was selected");
   return;  // exit the event handler
}

// otherwise, selectedValue might be TX if Texas was selected
```

- **Another trick starts with using the `selectedIndex` property to determine the index number of the selected option**
  - 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 `selectedIndex` to look up the actual item in the `select` element's `options` collection**

**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);
}
```

# 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**

NOTE: We are showing the options in HTML rather dynamically loading the dropdown to focus more on what we are illustrating. You could still load the dropdown using JavaScript.

**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;
```

# 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 = "WA";

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

- **Another trick is to 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;
   }
}
```

# 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="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;
```

# onchange Event

- **The onchange event occurs when the value of a <select> element has been changed**
  - You can handle this event 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
}
```

# Exercises

EXERCISE 1

Continue working in the Football project.

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.

EXERCISE 2

Continue working in the Football project.

Now, place your form elements in a `form` tag. Change the button to a `submit` button. Associate your event handler with the form's submit event. Don't forget to return false at the end of the event hander.

Test your page. It should still work.

EXERCISE 3

Continue working in the Football project.

Write code to add an option programmatically to the team dropdown with the text "Select a team" and the value "" before you load the actual teams.

You will need to make some changes to your event handler -- if the user selects the "Denver Broncos" and clicks the button, then goes back and selects "Select a team", you will need to clear the text in the paragraph.

(Optional) EXERCISE 4

Continue working in the Football project.

Find pictures of each team and show that picture beside the paragraph when the team is selected.

# Mini-Project

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

Design a web page that allows a user to select a category of menu items from a dropdown. The page will respond by displaying the category options in a listbox (Remember, a listbox is a `select` element that has the size attribute set)

Here is an example of the data structure you might use:

```
let menu = {
   drinks : [
      "Water", "Tea", "Sweet Tea",
      "Coke", "Dr. Pepper", "Sprite"
    ],
   entrees : [
      "Hamburger w/ Fries",
      "Grilled Cheese w/ Tater Tots",
      "Grilled Chicken w/ Veggies",
      "Chicken Fried Steak w/ Mashed Potatoes",
      "Fried Shrimp w/ Coleslaw",
      "Veggie Plate"
   ],
   desserts: [
      "Cheesecake", "Chocolate Cake", "Snickerdoodle Cookie"
   ]
};
```

Create a category dropdown that contains the options: "Drinks", "Entrees" and "Desserts". You can load the options statically in the HTML. Add a listbox below the dropdown.

NOTE: There will be NO button in this example. The listbox will be updated with the items that match the selected category when the change event on the dropdown occurs.

Write JavaScript to handle the change event for the category dropdown list. In the event handler, clear any items in the listbox and then re-load it with the matching items from the selected category.

Test your page. DON'T FORGET TO commit and push.

# Module 4

# Odds and Ends

# Section 4–1

# JavaScript and Truthy/Falsy Values

# 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

}
```

- **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`

- **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)

# 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
```

# 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!