Learn to Code

JavaScript

Workbook 3

Version 5.0 Y

author: Dana L. Wyatt, Ph.D.

Table of Contents

Module 1

Modularizing Code using Functions

Section 1–1

Modularizing Code

Modularizing Code

- **When you write code all day, every day, you start to see patterns**
  - You find yourself writing the same code over and over
- **At that point, you think: "Is there a better way?"**
  - And of course, the answer is usually "yes!"
- **JavaScript allows you to group together lines of code into a function to make it reusable**
  - By giving the function a name, you can reuse that code over and over
- **Functions also make it easy for teams of programmers to work on the same project**
  - Perhaps one programmer might write a function containing a complicated set of code while another is trying to get the login page to work
  - You can also reuse functions that other people have written

JavaScript Functions

- **As we saw in the last module, a JavaScript function is a block of code that is designed to perform a specific task when called**

- **In its simplest form, a JavaScript function is defined using:**
  - the keyword `function` keyword
  - the name of the function
  - a set of parentheses `( )`.
  - a set of curly brackets `{ }` that contains the body of the function

Syntax

```
function function-name() {
   // code to executed
}
```

- **You call the function by using its name, followed by parenthesis**

Example

```
// This is the function
function displayGreeting() {
   let message = "Hello world!";
   console.log(message);
}

// elsewhere...
// This is a call to the function
displayGreeting();
```

- **There is no limit to the number of lines of code in a function**

Example

```
function displayWeekdayMottos() {
   console.log("Make it Happen Monday!")
   console.log("Tackle it Tuesday!")
   console.log("Finish it Friday!")
}

displayWeekdayMottos();
```

Exercises

Create a new folder in your `LearnToCode` repo named `Workbook3`.

Now create a GitHub repo named `WB3-exercises`. Clone it under `LearnToCode\Workbook3`.

Finally, create a subfolder named `Functions`. These exercises should be placed there.

EXERCISE 1

Write a script named `simple_functions.js`. In it, define three functions:

1. The first should be named `favoriteThings()`. In it, use a `console.log()` statement to display 3 lines listing your name, your favorite movie, and your favorite musician/band.
2. The second should be named `whyImHere()`. In it, display a sentence that describes why you joined this "Learning to Code" program.
3. The last should be named `favoritePlace()`. In it, display a sentence that describes one of your favorite places to visit and why.

Below where you coded the three functions, call each of your functions.

DON'T FORGET to commit and push your repo.

Section 1–2

Passing Parameters to Functions

Parameters

- **Sometimes, you design functions to work on values that are passed to them**
  - These values are called parameters and are listed in the parenthesis after the function name

Syntax

```
function name(parameter1, parameter2 /* could be more */) {
   // code to executed
}
```

- **When the function is called, values are passed to the parameters**
  - The value passed is called the argument
  - The variable that "catches" the value in the function is called the parameter

Example

```
function display(msg) {    // msg is the parameter
   console.log(msg)
}

// The literal "Howdy" is the argument
display("Howdy");

// The literal "Ya'll come back..." is the argument
display("Ya'll come back now, ya hear!");

// x is the argument
let x = "Tackle it Tuesday";
display(x);
```

- **When there is more than one parameter, arguments are copies to parameters in the order they are listed**
  - Argument 1 is copied to parameter 1
  - Argument 2 is copied to parameter 2 …

Example

In this example, `name` and `age` are parameters. They received the values in the arguments `someName` and `someAge` that are passed to the function.

```
function displayNameAndAge(name, age) {
   let message = name + " is " + age;
   console.log(message);
}

// elsewhere

let someName = "Ezra";
let someAge = 17;

displayNameAndAge(someName, someAge);

someName = "Ian";
someAge = 16;

displayNameAndAge(someName, someAge);
```

- **REMEMBER: The argument names might, or might not, be the same as the parameters… the arguments are passed *positionally* to the parameters**

Built-in Functions

- **We've already seen some functions that are built-in to JavaScript**
  - They are already coded and all you have to do is call them
- **They typically have parameters that contain the values they work with**
- **We've already seen several built-in functions**
  - `parseInt()`
  - `parseFloat()`
  - `Number()`

Example

```
let x = "123";
let num = parseInt(x);
```

- **We've also seen how the `Math` object provides functions**

Example

```
let x = 150;
let y = 200;
let z = Math.abs(x - y);
```

- **We will eventually understand why the syntax between the `parseInt()` and `Math.abs()` is different**
  - For now, just find an example of the built-in function you need and pattern your use of the function after the example you find!

- **In class exercise:**
  - Google "Finding the absolute value of a number in JavaScript"
  - Look at one or two example
  - Now, how would you take the absolute value of a variable called `distance` and store it in a variable called `positiveDistance`?

_____

Exercises

Continue working in the `Functions` folder.

EXERCISE 1

Write a script named `more_functions.js`. In it, define three functions:

1. The first should be named `displayMailingLabel()`. It will accept five parameters: `name`, `address`, `city`, `state` and `zip`. In it, use `console.log()` to format and display the data like you would on an address label.
2. The second should be named `addNumbers()`. It will accept two parameters: `num1` and `num2`. Add the parameters together and display the result using the following format:

```
someNumber + someNumber = someNumber
```

3. The last should be named `displayReceipt()`. It will accept two parameters: `totalDue` and `amountPaid`. Compute and display the change due using the following format:

```
Total Due: $someNumber
Amount Paid: $someNumber

Change Due: $someNumber
```

   If the amount paid is less than the total due, display a message indicating how much more needs to be paid.

Now that you've defined your functions:

- call `displayMailingLabel()` twice with data for two different people
- call `addNumbers()` twice with different numbers
- call `displayReceipt()` three times. In one call, you should overpay the bill, in another you should pay the bill exactly, and in the last, you should underpay the bill.

DON'T FORGET to commit and push your repo.

Section 1–3

Functions Can Return Values

Returning Values

- **Functions sometimes compute results or find answers to questions and need to return the value back to the caller**
  - They do this using a `return` statement

- **When a value is returned, the calling code must "catch" the returned value and then use it as needed**

Example

```
function getNumGrandkids() {
   // in a better example, we might look this up in a database
   return 6;
}

let num = getNumGrandKids();
console.log(num);
```

- **Instead of catching the returned value in a variable, you can also use it in an expression**

Example

```
console.log("# grandkids is " + getNumGrandKids());
```

Example

```
function add(num1, num2) {
   let sum = num1 + num2;
   return sum;
}

// elsewhere

let a = 10, b = 20, c;
c = add(a, b);
console.log("The sum of " + a + " and " + b + " is " + c);
```

- **Often, functions have to make decisions as to what value they return**

Example

```
function getNumGrandKids(name) {  // name is the parameter
   let num = 0;

   if (name == "Dana") {
      num = 6;
   }
   else (if name == "Karla") {
      num = 3;
   }
   else if (name == "Leslye") {
      num = 2;
   }

   return num;
}

let num = getNumGrandKids("Dana");  // "Dana" is argument
console.log(num);

num = getNumGrandKids("Karla");
console.log(num);
```

Exercises

Continue working in the `Functions` folder.

EXERCISE 1

Create a file called `f_to_c.js`. In it, create a function called `convertFtoC` that accepts the temperature in Fahrenheit and returns the Celsius equivalent.

You will be able to call the function using code similar to that below:

```
let currentTemp = 92;
let celsiusTemp = convertFtoC(currentTemp);
```

Use the function to convert the following Fahrenheit values and display them in Celsius:

. 212, 90, 72, 32, 0 and -40

EXERCISE 2

Create a file called `c_to_f.js`. In it, create a function called `convertCtoF` that accepts the temperature in Celsius and returns the Fahrenheit equivalent.

Use the function to convert the following Fahrenheit values and display them in Celsius:

. 100, 45, 19, 0, -7 and -40

EXERCISE 3

Create a file called `tax_functions.js`. In it, you will create three functions:

`getSocSecTax()`	it accepts a gross pay and returns the Social Security tax on that amount. Assume a tax rate of 6.2%
`getMedicareTax()`	it accepts a gross pay and returns the Medicare tax on that amount. Assume a tax rate of 1.45%

`getFederalTax()` it accepts a gross pay and withholding code and returns the federal tax withholding on that amount. Use the tax table below.

You will need to use an `if / else` statement! The case for a withholding code of 4 or more is tricky!

Withholding Code	Tax Rate
0	23%
1	21%
2	19.5%
3	18.5%
4+	18% less 0.5% for each withholding over 4 (ex: code 6 - rate .18-(2 x .005) = .17 (or 17%)

Call the functions 3 times with each following people and display the results.

Person 1:	gross pay $750	withholding code 0
Person 2:	gross pay $1550	withholding code 2
Person 3:	gross pay $1100	withholding code 6

DON'T FORGET to commit and push your repo.

Section 1–4

Scoping Rules

Scoping

- **Scope is a word that is used to describe how long a variable lives and where it can be accessed**

- **Until ES6 (or ECMAScript 2015), there were two types of scope**
  - Global scope
  - Function scope

- **A variable define outside of any function has *global scope***
  - That means it can be accessed from anywhere in JavaScript

Example

```
let count = 0;

function increment() {
   count = count + 1;
}
```

- **A variable define inside of a function has *function scope***
  - That means it can only be accessed from within that function

Example

```
function showName() {
   let name = "Brittany";
   console.log(name);
}

// cannot access the variable name here
```

- **If you don't declare a variable in a function, it tries to use a globally scoped one of the same name**

Example

```
let number = 5;

function test() {
    number = 6;
    console.log("++ " + number);
}

console.log("** " + number);
test();
console.log("** " + number);

OUTPUT
** 5
++ 6
** 6
```

- **JavaScript maintains separate areas of memory for global and function scoped variables so you don't have to worry about name collisions**

Example

```
let number = 5;

function test() {
    let number = 6;
    console.log("++ " + number);
}

console.log("** " + number);
test();
console.log("** " + number);

OUTPUT
** 5
++ 6
** 5
```

`var` and `"use strict"`

- **As we mentioned earlier, by default JavaScript does not actually require you to use `var` to declare variables**
  - This means a misspelled variable name creates a new global variable

Example

```
var number = 5;
...
numebr = number + 1;
   // number is still 5
   // numebr is 6
```

- **Adding the `'use strict'` directive that was introduced in ES5 changes that rule**
  - Any variable not defined using `var` generates an error

Example

```
"use strict";

var number = 5;
...
numebr = number + 1;       // error: numebr is not defined
```

Re-declaring Variables using `var`

- **JavaScript allows you to re-declare a variable using `var`**
  - That means a new declaration "overwrites" the original one

Example

```
"use strict";

var i = 5;

...

var i = 10;

...

console.log(i);       // i is 10
```

- **This is NOT the case if you had used `let`**
  - This is one reason we like let! It keeps us from accidently re-declaring a variable

Example

```
"use strict";

let i = 5;

...

let i = 10;           // ERROR!!
```

Hoisting `var` Variables

- **By default, JavaScript variables defined with the keyword `var` are "hoisted" to the top of the scope they are defined within**

- **Hoisting means that the JavaScript engine moves all variable declarations to the top of the current scope**
  - That is, to the top of the script or the function

- **This means you can use a variable before it is declared**

Example

```
name = "Ezra";

var name;              // weird, huh?
```

- **However, initializations are NOT hoisted**
  - This means you can have some unusual behaviors

Example

```
var num1 = 5;

var answer = num1 + num2;
console.log(answer);       // displays NaN

var num2 = 7;              // because num2 is initialized here
```

New ES6 Block Scope Options

- **ES6 introduced two new keywords (`let` and `const`) that add additional scoping rules**
  - `let` provides block scope
  - `const` provides block scoped constants
- **Variables defined with `let` have block scope**
  - A block is defined using `{ }`
  - Any variable defined using `let` cannot be accessed outside of the block it is defined in

Example

```
function addNums(a, b, c) {
   let sum;
   if (a < 0) {
      sum = Math.abs(a) + Math.abs(b) + Math.abs(c);
   }
   else {
      sum = a + b + c;
   }
   console.log(sum);            // this displays 11
}

let x = addNums(-1, 3, 7)
```

- **But because the scope of a `let` variable is the block it is defined in, the following example would fail**

Example

```
function addNums(a, b, c) {
   if (a < 0) {
      let sum = Math.abs(a) + Math.abs(b) + Math.abs(c);
   }
   else {
      let sum = a + b + c;
   }
   console.log(sum);            // this generates an error
}

let x = addNums(-1, 3, 7)
```

- **If `sum` was declared using the keyword `var` in the above example, the code would have worked because of hoisting**
  - NOTE: This doesn't mean using `let` is bad! In fact, it is good.

More About `let`

- **Variables defined with the keyword `let` are NOT hoisted to the top**
  - This means if you use the variable before it is declared, JavaScript will generate an error

Example

```
name = "Ezra";  // ERROR

let name;
```

  - NOTE: This doesn't mean using `let` is bad; it keeps you from having the weird initialization problem we saw earlier

- **Variables defined using `let` cannot be re-declared**

Example

We will talk about the `for` loop in an shortly.

```
let i = 5;

for (let i = 0; i < 10; i++) {  // this is an error
  // statement(s)
}
```

Constants

- **If you define your variable with `const`, the variable cannot be changed once it has been initialized**

  Example

```
const TAXRATE = .0825;

TAXRATE = .0875;          // This will generate an error
```

- **Like `let`, variables declared with `const` have block scope**

- **`const` variables must be assigned a value when they are declared**

  – Because you can't give them a value later!

  Example

```
const TAXRATE;                  // This is an error
```

- **Getting into the habit of using `const` for things that won't change is a good practice!**

- **Best practice: try to use `let` instead of `var` if your browsers will support it**

  – Also, make sure to add `"use strict"`

Exercises

EXERCISE 1

Answer the following questions.

#1 What is the output from running this code?

```
"use strict";
function test1() {
  let a = 10;
  if (a > 5) {
    a = 7;
  }
  console.log("a = " + a);
}

test1();
```

#2 What is the output from running this code?

```
"use strict";
function test2A() {
  if (1 == 1) {
    var a = 5;
  }
  console.log("a = " + a);
}

test2A();
```

What about this code?

```
"use strict";
function test2B() {
  if (1 == 1) {
    let a = 5;
  }
  console.log("a = " + a);
}

test2B();
```

#3 What is the output from running this code?

```
"use strict";
let a = 4;
function test3() {
  a = 3;
  console.log("a = " + a);
}

test3();
console.log("a = " + a);
```

#4 What is the output from running this code?

```
"use strict";
let a = 4;
function test4() {
  let a = 7;
  console.log("a = " + a);
}

test4();
console.log("a = " + a);
```

#5 CHALLENGE! What is the output from running this code?

```
"use strict";
let a = 4;
function test5() {
    a = 7;
    function again() {
        let a = 8;
        console.log("a = " + a);
    }
    again();
    console.log("a = " + a);
}

test5();
console.log("a = " + a);
```

(see next page for last one)

#6 CHALLENGE! What is the output from this function?

```
"use strict";
let a = 4;
function test6() {
    let a = 7;
    function again() {
        let a = 8;
        console.log("a = " + a);
    }
    again();
    console.log("a = " + a);
}

test6();
console.log("a = " + a);
```

Module 2

Working with Strings and Dates

Section 2–1

Using Strings

Strings

- **JavaScript strings store text**
  - The actual string can have zero or more characters written inside quotes

Example

```
let name = "Dana L. Wyatt";
let favoriteQuote = "";
```

- **JavaScript allows you to use single or double quotes**

Example

```
let name = 'Dana L. Wyatt';
let favoriteQuote = '';
```

- **Best practices says it doesn't matter which you use as long as you are consistent**

- **However, if you need to embed a single or double quote inside of the string, you may need a combination of each**

Example

```
let quote1 = "In Texas, we use the word ya'll often";
let quote2 = 'Big Tex (at the State Fair) says "Howdy"';
```

- **To find the length of a string, use the `length` property**

Example

```
let name = "Mark Westly";
let nameLength = name.length;

console.log(name + " is " + nameLength + " characters long");

    // displays: Mark Westly is 11 characters long
```

Strings and Escape Characters

- **Sometimes, you must use escape characters to embed special characters into a string**
  - **\"** is the double quote character
  - **\'** is the single quote character
  - **** is the backslash character

Example

```
let quote1 = 'In Texas, we use the word ya\'ll often';
let quote2 = "Big Tex (at the State Fair) says \"Howdy\"";

let quote3 = "\\Heading/";
   /*   quote3 holds \Heading/   */
```

Long Strings Literals

- **If a JavaScript statement does not fit on one line, break it after an operator**

Example

```
let message =
   "The best movie in the world is the first Mamma Mia!";
```

- **Sometimes, you have to build strings by concatenating shorter strings together**

Example

```
let message = "The best movie in the world " +
              "is the first Mamma Mia!";
```

Example

```
let movie = "Mamma Mia"
let message = "The best movie in the world " +
              "is the " + movie + "!";
```

Working with Case and Excess Whitespace

- You can convert a string to upper case using **`toUpperCase()`**

- You can convert a string to lower case using **`toLowerCase()`:**

Example

```
let s1 = "Hello World!";
let s2 = s1.toUpperCase();    // HELLO WORLD!
let s3 = s1.toLowerCase();    // hello world!
```

- The **`trim()`** method removes whitespace from each side of a string

Example

```
let s1 = "         Hello World!          ";
let s2 = s1.trim();
     /* contains "Hello World!" */
```

Searching Strings

- **You can search a string to find the location of a string in a string**
  - `indexOf()` returns the position of the first occurrence of specified text
    - The position is zero-based, which means if the first character matches, it returns 0 (instead of 1)

Example

```
let name = "Betty Jo Smalltree";
let pos = name.indexOf(" ");
    // finds the position of the first space (ex: 5)
```

  - `lastIndexOf()` returns the position of the last occurrence of specified text

Example

```
let name = "Betty Jo Smalltree";
let pos = name.lastIndexOf(" ");
    // finds the position of the last space (ex: 8)
```

- **If the string you are searching for is not found, both functions return -1**

Extracting Substrings

- **There are several ways to extract a part of a string, including:**
  - `substring(start, end)`
  - `slice(start, end)`
  - `substr(start, length)` -- note: legacy function so avoid if possible

- **`substring()` extracts part of a string and returns it in a new string**
  - The ending position is NOT included in the extracted string
  - If you omit the second argument, `substring()` will extract the rest of the string

Example

```
let name = "Betty Jo Smalltree";
let first = name.substring(0, 5);   // "Betty"
let last = name.substring(9);       // "Smalltree"
```

- **`slice()` is similar to `substring()` except that it also supports negative indexes**

Example

```
let name = "Betty Smalltree";
let first = name.slice(0, 5);   // first contains "Betty"
let last = name.slice(6, 15);   // last contains "Smalltree"
```

  - If the parameter is negative, the position is counted from the end of the string

Example

```
let name = "Betty Jo Smalltree";
let mid = name.slice(-12, -10);     // mid contains Jo

// you could have also done the same thing from the beginning

let name = "Betty Jo Smalltree";
let mid = name.slice(6, 8);         // mid contains Jo
```

Examples: Finding Substrings

- **Sometimes, you know exactly which part of a string you need to find**
  - US Social Security numbers have the format xxx-xx-xxxx

Example

```
let ssn = "111-22-3333";

let first3 = ssn.substring(0, 3);
let second2 = ssn.substring(4, 6);
let last4 = ssn.substring(7, 11);

let ssnWithoutDashes = first3 + second2 + last4;
```

- **When you don't know where the substrings begin and end, you must look for delimiters**

Example

```
let partCode = "275656543-large";
let partNum, size;

let dashPos = partCode.indexOf("-");
partNum = partCode.substring(0, dashPos);
size = partCode.substring(dashPos + 1);
```

Template Strings (aka String Interpolation)

- **ES6 introduced template literals allowing you to embed expressions into your string**
  - This feature is sometimes called string interpolation
- **String templates use the back tick (`) character to enclose the string**
- **By using these string templates, you can avoid concatenations**

Example

```
// using concatenation

let name = "Pursalane";
let age = 10;

let msg = "Student: " + name + " (age: " + age + ")";
```

Example

This examples inserts the value of the `name` and `age` variables where the `${ }` placeholder is located

```
// using template strings

let name = "Pursalane";
let age = 10;

let msg = `Student: ${name} (age: ${age})`;
```

- **However, this is not currently supported by IE**

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals

Example: Stripping Delimiters from a String

- **What if you needed to strip out delimiters and wanted to encapsulate that logic in a function?**

Example

```
function stripHyphensFromSSN(ssn) {

  let firstPart = ssn.substring(0, 3);
  let secondPart = ssn.substring(4, 6);
  let thirdPart = ssn.substring(7);

  let ssnDigitString = firstPart + secondPart + thirdPart;

  return ssnDigitString;
}

// elsewhere
let ssn = "123-45-6789";
let ssnWithoutHyphens = stripHyphensFromSSN(ssn);
console.log(ssnWithoutHyphens);
```

- **Can you envision the logic to strip out the punctuation from a phone number formatted like (817)555-5555?**

- **Functions can only return a single value. How could you code `getProductNumber()` and `getProductSize()` functions?**

Accessing Characters in a String

- **The `charAt()` method can be used to get the character at a specified position in the string**

  Example

```
let myString = "How now brown cow";
console.log(myString.charAt(14));        // displays 'c'
```

- **ES5 introduced the ability to use `[ ]` to access a character in a string**
  - Like `charAt()`, it is read only

  Example

```
let myString = "How now brown cow";
console.log(myString[14]);           // displays 'c'
```

  - It makes strings look like arrays (but they are not arrays)

- **If no character is found in the examples above:**
  - `charAt()` returns the empty string
  - `[ ]` returns `undefined`

- **If you want interact with a string as an array, you can convert it to an array**
  - We will discuss arrays shortly

Exercises

Create a subfolder under `WB3-exercises` named `Strings`. The exercises in this section should be placed there.

EXERCISE 1

Create a new script file and name it `parsing.js`. Within it, declare a variable that contains a first and last name:

```
let name = "Brenda Kaye";
```

Write code to find the space using `indexOf()` and then extract and display the first and last name. Example output might be:

```
Name: Brenda Kaye
First name: Brenda
Last name: Kaye
```

Once it works, move your code into a function called `parseAndDisplayName()` and call it using the following:

```
parseAndDisplayName("Brenda Kaye");
parseAndDisplayName("Ian Auston");
parseAndDisplayName("Siddalee Grace");
```

(Challenge) EXERCISE 2

Copy your code to a new script named `advanced_parsing.js`

Modify the function to work with one, two, or three word names.

```
parseAndDisplayName("Cher");
parseAndDisplayName("Brenda Kaye");
parseAndDisplayName("Dana Lynn Wyatt");
```

Break into groups of 2-3 students to design your algorithm. Call over your instructor if you need ideas! Then code it.

The output of the code below should be:

```
Name: Cher
Only name: Cher
Name: Brenda Kaye
First name: Brenda
Last name: Kaye
Name: Dana Lynn Wyatt
First name: Dana
Middle name: Lynn
Last name: Wyatt
```

EXERCISE 3

Create a new script file and name it `part_codes.js`. Within it, create a script that examines and parses a part code in the following format:

`supplierCode``:`*`productNumber-size`*

So for example,

```
ACME:123-L     the large (L) part 123 is supplied by ACME
DI:12345-M     the medium (M) part 12345 is supplied by DI
ACE:1-12       the size 12 part 1 is supplied by ACE
```

Sketch the functions defined below on a piece of paper:

```
function getSupplier(code)
  // that returns all characters before the :

function getProductNumber(code)
  // that returns all characters between the : and -

function getSize(code)
  // that returns all characters after the -
```

Perform a code review with a classmate. Then, code the script.

Finally, declare variables that hold the three part codes at the top of this exercise. call all three functions for each part code and display the results.

DON'T FORGET TO commit and push your repo.

Converting a String to an Array

- **JavaScript lets you convert a string to an array using `split()`**
  - We will learn about arrays in more detail later this week

Example

```
let inputs = "San Francisco,Dallas,Atlanta,Hartford";
let array = inputs.split(",");

// The array resembles the following
//     position 0  -  value "San Francisco"
//     position 1  -  value "Dallas"
//     position 2  -  value "Atlanta"
//     position 3  -  value "Hartford"

for (let i = 0; i < 4; i++) {
   console.log(array[i]);
}
```

  - We will learn more about arrays next week!

Exercises

Continue working in the `Strings` folder.

EXERCISE 1

There are many string methods we haven't looked at yet. For this exercise, you need to go to `www.w3schools.com` and find their JavaScript section.

What is the URL you found it at?

_____

On the left, you should see various topics. Click on the link for "String Methods".

Find the example for **replacing string content**. Study it.

Now code a script named `replace_strings.js` that contains the following variable:

```
let message = "Our corporate offices are located in Dallas";
```

Use the technique you found in the help to replace the word "Dallas" with "Austin". Print out your new string.

Now try the technique that finds the string using a case insensitive match. It uses something called regular expressions that we will eventually discuss in more detail. Did you get it to work? Make sure to use "DALLAS" or "dallas" when you are specifying the string to replace.

Section 2–2

Using Dates

JavaScript Dates

- **The `Date` object is used to work with dates and times**

- **JavaScript stores date/time information as the number of milliseconds since January 1, 1970**
  - Zero time is January 01, 1970 00:00:00 UTC

- **However, you can ask JavaScript to return date information as month, day, year, weekday, hour, minute, and second**
  - Be careful! JavaScript uses ***zero-based numbers*** for both months and weekdays
    - That means the month number 0 is January and 11 is December
    - That also means the weekday number 0 is Sunday and 6 is Saturday

- **To get the current date/time from the browser, use `new Date()`**

Example

```
let d = new Date();
console.log(d);

OUTPUT IN CODERUNNER
2022-05-29T12:50:04.484Z

OUTPUT IN A BROWSER
Fri May 29 2022 07:50:04 GMT-0500 (Central Daylight Time)
```

Displaying Dates

- **You can turn a `Date` object into a string to display it consistently using several different methods**
  - The `toString()` method will show date/time in a text string and includes the time zone

Example

```
let d = new Date();
console.log(d.toString());

OUTPUT
Fri May 29 2022 07:50:04 GMT-0500 (Central Daylight Time)
```

  - The `toUTCString()` method will show date/time in a text string as a UTC time (aka GMT time)
    - More on UTC time in a few minutes

Example

```
let d = new Date();
console.log(d.toUTCString());

OUTPUT
Sat, 29 May 2022 12:50:04 GMT
   // 12:50 GMT is 7:50 AM in Texas where this code ran!
```

- **If you want only the date, `toDateString()` returns a only the date string**

Example

```
let d = new Date();
console.log(d.toDateString());

OUTPUT
Sat, 29 May 2022
```

- **There are three functions that take a `Date` object and convert it to a string according to the current locale setting on the user's machine**

Example

```
let d = new Date();

console.log(d.toLocaleDateString());
console.log(d.toLocaleTimeString());
console.log(d.toLocaleString());

OUTPUT
5/29/2022
07:50:04 AM
5/29/2022, 07:50:04 AM
```

Creating Dates in Different Ways

- **You can create an instance of a `Date` by passing numeric values to the `Date` constructor**
  - Number values, in order are: year, month (0-based), day, hour, minute, second, millisecond

Example

```
let date1 = new Date(1934, 7, 12);
     // August 12, 1934
let date2 = new Date(1926, 6, 28, 10, 2, 0);
     // July 28, 1926  10:02:00 AM
```

- **You can create a date using the ISO 8601 format**
  - ISO dates can be written with or without hours, minutes, and seconds (YYYY-MM-DDTHH:MM:SSZ)
    - MM is 1-based
    - The Z at the end specifies UTC time
    - If you omit time zone, the result is converted to the browser's time zone

Example

```
let d = new Date("2022-04-01T12:00:00Z");
```

- **There other ways dates can be created, including some shown below**
  - Long dates can be written as "MMM DD YYYY"
  - Short dates can be written using "MM/DD/YYYY"

– Note: Some browsers may generate an error if you use a single digit for the month or day

Example

```
// When date and time are specified, results are obvious
let d1 = new Date("May 20, 1990 03:20:00");
    // May 20, 1990

// When only a date is specified, time defaults to midnight
let d2 = new Date("08/05/1986");
    // August 5, 1986

let d3 = new Date("Sep 05 1958");
    // September 5, 1958

let d4 = new Date("12 December 1982");
    // December 12, 1992
```

- **But be careful! Converting a string to a date *sometimes* results in a value that is 1 day behind!**

Example

```
let d5 = new Date("2004-06-25");
    // June 24, 2004   <--------------- issue!
```

- **Why?**
  - It has to do with the specific string format it is converting from
    - *AND* the difference between YOUR computer and UTC time

Why a JavaScript Date Might Be One Day Off

- **What happens when we create a date using the string format YYYY-MM-DD?**

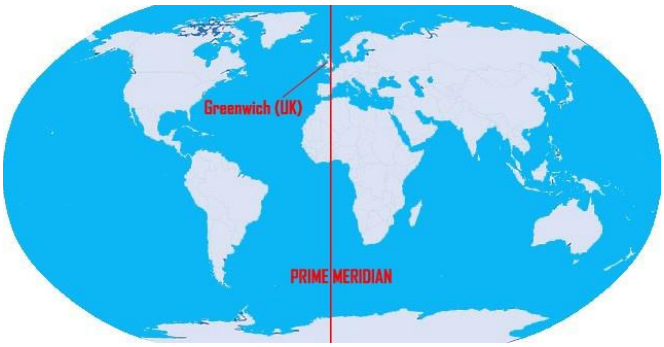
Example

```
// what if I create this right now
let d = new Date("2023-05-11");

console.log(d.toDateString());
   // displays Wed May 10 2023

console.log(d.toISOString());
   // displays 2023-05-11T00:00:00Z
```

- **The key to understanding this is time zones!**
  - In order to have an understanding of a "moment in time" be consistent, Greenwich Mean Time (GMT) was established in 1847



  - Date/times could be established as an offset of a GMT time
  - In 1972, Universal Coordinated Time (UTC) was established as a slightly more consistent formulation of calculating time that included "leap seconds"

- For a fun discussion, see: https://www.zachgollwitzer.com/posts/2020/js-dates/

- **Texas, in the summer, is in the Central Daylight time zone and is 5 hours behind GMT (Greenwich Mean Time)**

Example

```
// When I create the date this way, it thinks I'm in GMT
// and it uses a 1-based month
let d = new Date("2023-05-11");

// But when I print it out, the display adjusts for my
// local time
console.log(d.toString());
   // displays
   // Wed May 10 2023 19:00:00 GMT-0500 (Central Daylight Time)
```

- **But now it gets more complicated**
  - When you create the date using the date constructor with three separate values…

Example

```
// When I create the date this way, it uses my local time zone
// AND it also uses a 0-based month
let d = new Date(2023, 5, 11);

console.log(d.toString());
   // displays
   // Sun Jun 11 2023 00:00:00 GMT-0500 (Central Daylight Time)
```

  - When you create the date using the date constructor with three separate values…

Example

```
// When I create the date this way, it uses my local time zone
let d = new Date("05-11-2021");

console.log(d.toString());
   // displays
   // Thu May 11 2023 00:00:00 GMT-0500 (Central Daylight Time)
```

Example

```
// When I create the date this way, it uses my local time zone
let d = new Date("05/11/2023");

console.log(d.toString());
   // displays
   // Thu May 11 2023 00:00:00 GMT-0500 (Central Daylight Time)
```

- **So, you will need to make sure you are passing a string to the `Date` constructor that won't cause any off by 1 errors!**

- **Will this impact you? --- YES!**
  - When you use the `<input type="date">` and fetch its value, it comes back in the form "year-mm-dd" (ex: "`2023-05-11`")

Exercise

Create a folder under `WB3-exercises` named `Dates`. Under that, a folder named `OffBy1Website`. This exercise goes there.

EXERCISE 1

Create an `index.html` page and add:

1. an input whose type is date
2. a button
3. a paragraph to display a message

In your `index.js`, handle the button click. In the event handler, get the value in the input box and display it as a string in the paragraph.

Now, change the code to convert the string to a date and then display it in the paragraph using `toString()`

DON'T FORGET TO commit and push your repo.

Getting Access to Date Fields

- **There are methods that allow you to reach into a `Date` object and extract specific values**
  - `getMonth()` Get the month (0-11)
  - `getDate()` Get the day (1-31)
  - `getFullYear()` Get the year (yyyy)
  - `getDay()` Get the weekday (0-6)
  - `getHours()` Get the hour (0-23)
  - `getMinutes()` Get the minute (0-59)
  - `getSeconds()` Get the second (0-59)

Example

In this example, the month was June

```
let d = new Date();
console.log(d.getMonth() + 1);

OUTPUT
6
```

Getting a Month Name

- **If you want to obtain the month name, you need to look up the spelling of the month in an array**

Example

```
const months = ["January", "February", "March", "April",
                "May", "June", "July", "August", "September",
                "October", "November", "December"];
let d = new Date();
let monthName = months[d.getMonth()];
```

Setting Date Fields

- **There are similar methods that allow you to reach into a Date object and set specific values**
  - `setMonth()` Set the month (0-11)
  - `setDate()` Set the day (1-31)
  - `setFullYear()` Set the year (yyyy)
  - `setHours()` Set the hour (0-23)
  - `setMinutes()` Set the minute (0-59)
  - `setSeconds()` Set the second (0-59)

Example

The code below sets the date to FEBRUARY 9, 1963

```
let d = new Date();
d.setMonth(1);
d.setDate(9);
d.setFullYear(1963);
```

Example

The code below determines when one year from today will be

```
let d = new Date();
d.setFullYear(d.getYear() + 1);
```

Exercises

Create a subfolder under `Dates` named `DateScripts.` These exercises should be placed there.

EXERCISE 1

Create a script named `date_to_string.js` that grabs the current date/time. Then, display it using:

1. `toString()`
2. `toDateString()`
3. `toUTCString()`
4. `toLocaleString()`

EXERCISE 2

Create a script named `more_date_conversions.js` that creates date objects for the following important dates in your life. (Improvise if you don't know the actual date) Then, display each using `toLocaleString()`

1. Your birthday using the form "MM/DD/YYYY"
2. Your mom's birthday using the form "MMM DD YYYY"
3. Your dad's birthday using `new Date(yyyy, m, d)`

EXERCISE 3

Create a script named `custom_date_format.js` that grabs the current date/time. Then, display it using the following format:

day-month-year (*weekdayame*)

example: 27-July-2022 (Monday)

You will need to use the `getXXX()` functions we just looked at, where XXX might be `getMonth()` or `getFullYear()`.

Parsing Dates

- **If you have a valid date string, you can use `Date.parse()` to convert it to milliseconds**
  - It returns the number of milliseconds between the date and January 1, 1970

Example

```
let milliSec = Date.parse("May 1, 2022");
```

- **You can then use the number of milliseconds to convert it to a `Date` object**

Example

```
let milliSec = Date.parse("May 1, 2022");
let d = new Date(milliSec);
```

Determining the Difference Between Two Dates

- **To determine the difference between two dates, you will need to:**
  - Convert each date to a millisecond timestamp using `getTime()`
    - Remember that dates are stored in JavaScript as the number of milliseconds that have elapsed since January 1, 1970
  - Subtract the two millisecond timestamps to determine the actual number of milliseconds between them
  - Turn that elapsed milliseconds back into days

Example

```
let date1 = new Date("January 15, 2022");
let date2 = new Date("March 15, 2022");

let msec_per_day = 1000 * 60 * 60 * 24;

let elapsedMilliseconds = date2.getTime() - date1.getTime();

let dayDiff = elapsedMilliseconds / msec_per_day;
let numDays = Math.round(dayDiff);

console.log("The number of days between dates is " + numDays);

OUTPUT
The number of days between dates is 59
```

Exercises

Continue working in `DateScripts`.

EXERCISE 1

Create a script named `date_difference.js` that grabs creates the two string variables below:

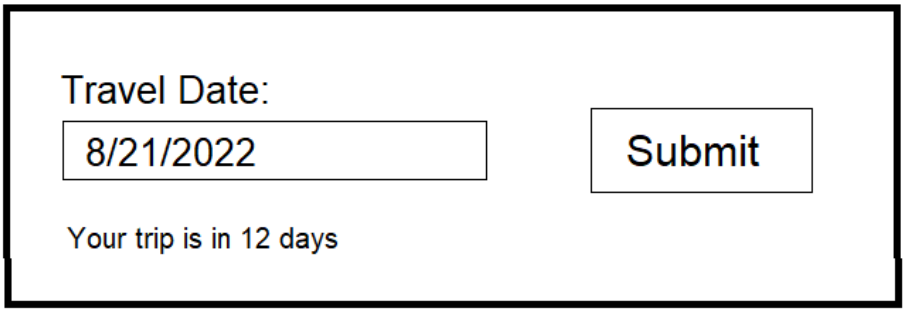
```
let startDate = "July 11, 2022";
let endDate = "November 11, 2022";
```

Now, write code to get and display the number of days between the two dates.

(Optional) EXERCISE 2

Design and implement a web page that tells you how far away (in days) you will leave for a trip! Be warry of the off by 1 issue!

Use an inline JavaScript script in this simple page!



DON'T FORGET TO commit and push your repo.

Module 3

Working with Forms - Part 2

Section 3–1

JavaScript and Anonymous Functions

JavaScript and Anonymous Functions

- **JavaScript supports a type of function called an anonymous function**
  - It is called that because it isn't given a name

- **You can use the anonymous function to assign event handling behavior to `window.onload`**

Example

HTML

```
<html>
<head>
   <title>Anonymous Functions</title>
</head>
<body>
   <input id="helloBtn" type="button" value="Say Hello" />
   <script src="scripts/index.js"></script>
</body>
</html>
```

JavaScript

```
"use strict";

window.onload = function() {
   const btn = document.getElementById("helloBtn");
   btn.onclick = sayHello;
};

function sayHello() {
   alert("Hello!");
}
```

- **This is a popular coding pattern used in JavaScript and we will use it in most pages we build during the course**

Exercises

EXERCISE 1

Create a subfolder under `WB3-exercises` named `AnonymousFunctionDemo`. This exercise should be placed there.

Create a page named `index.html` and enter the code on the previous page. Test your page.

Do you think it is easier to assign all of your event handlers when the page loads using this shortened syntax?

Section 3–2

Working with Check Boxes and Radio Buttons

Working with Checkboxes

- **Do you remember we create checkboxes using an `<input type="checkbox">` element?**

Example

```
<fieldset>
   <legend>Rental Car Options:<legend>
   <input type="checkbox" id="tollTag" name="tollTag"
       value="tolltag" checked> Electronic Toll Tag ($3.95/day
       plus tolls)<br>
   <input type="checkbox" id="gps" name="gps"
       value="gls"> GPS ($4.95/day)<br>
   <input type="checkbox" id="roadside" name="roadside"
       value="roadside"> Roadside Assistance ($2.95/day)<br>
</fieldset>
```

- **In JavaScript, you can determine whether a checkboxes is checked using the Boolean `checked` property**

Example

```
let extraPerDay = 0;

let tollTag = document.getElementById("tollTag").checked;
if (tollTag == true) {
   extraPerDay += 3.95;
}

let gps = document.getElementById("gps").checked;
if (gps == true) {
   extraPerDay += 4.95;
}

let roadside = document.getElementById("roadside").checked;
if (roadside == true) {
   extraPerDay += 2.95;
}
```

Shortcut for Checking for `true`

- **JavaScript makes it easy to check if a Boolean variable is true**

Example

```
// rather than writing
if (tollTag == true) {

}

// you can simply use the value of the boolean variable
// as the condition --
//     if the variable is true, the if is true
//     if the variable is false, the if is false
if (tollTag) {

}
```

Example

```
let extraPerDay = 0;

let tollTag = document.getElementById("tollTag").checked;
if (tollTag) {
   extraPerDay += 3.95;
}

let gps = document.getElementById("gps").checked;
if (gps) {
   extraPerDay += 4.95;
}

let roadside = document.getElementById("roadside").checked;
if (roadside) {
   extraPerDay += 2.95;
}
```

- **And if you only need to know if a checkbox is checked, you can completely omit putting the value of checked in a variable**

Example

```
let extraPerDay = 0;

if (document.getElementById("tollTag").checked) {
   extraPerDay += 3.95;
}

if (document.getElementById("gps").checked) {
   extraPerDay += 4.95;
}

if (document.getElementById("roadside").checked) {
   extraPerDay += 2.95;
}
```

  - However, if you are going to interact with the checkbox many times, getting the reference once and putting it in a variable is more efficient!

Events and Checkboxes

- **A checkbox supports the `onclick` event**

Example

In this example, when a user checks the checkbox that says they are a government employee, the roadside assistance checkbox is automatically checked

HTML

```
<input type="checkbox" id="govt" name="govt"
       value="govt"> Are you a US Government employee?<br>
```

JavaScript

```
window.onload = function() {
   let govtChkBox = document.getElementByid("govt");
   govtChkBox.onclick = onGovtChkBoxClicked;

   // other events handlers connected here
}

function onGovtChkBoxClicked() {
    let govt = document.getElementById("govt").checked;
    if (govt) {
       let roadsideChkBox =
                document.getElementById("roadside");
       roadsideChkBox.checked = true;
    }
}
```

Hiding and Showing Elements

- **So far, we've mostly seen how to get the contents of an input text field or determine if a checkbox is checked**

- **You can also use JavaScript to programmatically show or hide HTML elements**

Example

In this example, when a user checks the "ship to the same address" checkbox, the shipping address is hidden. If it is unchecked, the shipping address is shown.

HTML

```
<input type="checkbox" id="sameAddress" name="sameAddress"
       value="same" checked> Ship to the same address?<br>
<div id="shippingDiv">...</div>
```

JavaScript

```
window.onload = function() {
   let sameAddrChkBox = document.getElementById("sameAddress");
   sameAddrChkBox.onclick = onHideOrShowShippingDiv;

   // other events handlers connected here
}

function onHideOrShowShippingDiv() {
   let sameAddrChkBox = document.getElementById("sameAddress");
   let shippingDiv = document.getElementByid("shippingDiv");

   if (sameAddrChkBox.checked) {
      shippingDiv.style.display = "none";
   }
   else {
      shippingDiv.style.display = "block";
   }
}
```

Working with Radio Buttons

- **Radio buttons are similar to check boxes, except that only one in a group may be checked**
  - They must have the `name` in order to work correctly

Example

```
<fieldset>
   <legend>Select the policy type:<legend>
   <input type="radio" name="policy" id="auto"
          value="auto" checked> Auto<br>
   <input type="radio" name="policy" id="home"
          value="home"> Home<br>
   <input type="radio" name="policy" id="life"
          value="life"> Life<br>
</fieldset>
```

- **Radio buttons also have a Boolean `checked` property**

Example

```
let autoRadioBtn = document.getElementById("auto");
let homeRadioBtn = document.getElementById("home");

let basePremium = 0;
if (autoRadioBtn.checked) {
   basePremium = 175.00;
}
else if (homeRadioBtn.checked) {
   basePremium = 395.00;
}
else { // it must be life!
   basePremium = 225.00;
}
```

- **With both checkboxes and radio buttons, you can find the field and get its checked value in one statement if you like the way it reads**

Example

```
let basePremium = 0;
if (document.getElementById("auto").checked) {
   basePremium = 175.00;
}
else if (document.getElementById("home").checked) {
   basePremium = 395.00;
}
else {  // it must be life!
   basePremium = 225.00;
}
```

- **If you ever find a need to, you can programmatically set the `checked` property**

Example

```
document.getElementById("life").checked = true;
```

- **Radio buttons also support the `onclick` event**
  - Unlike checkboxes, if a radio button's click event is triggered, it is definitely clicked!

Using `querySelector()`

- **We know that you can find an HTML element on a page using `getElementById()`**

- **You can also find an HTML element using a CSS selector combined with the function `querySelector()`**

Example

`querySelector()` will select the first `input` elements with a `name` attribute of `policy` that is checked.

HTML

```
<fieldset>
   <legend>Select the policy type:<legend>
   <input type="radio" name="policy" id="auto"
          value="auto" checked> Auto<br>
   <input type="radio" name="policy" id="home"
          value="home"> Home<br>
   <input type="radio" name="policy" id="life"
          value="life"> Life<br>
</fieldset>
```

JavaScript

```
let selectedOption =
   document.querySelector("input[name='policy']:checked");

let basePremium = 0;
if (selectedOption.value == "auto") {
   basePremium = 175.00;
}
else if (selectedOption.value == "home") {
   basePremium = 395.00;
}
else {
   basePremium = 225.00;
}
```

Accessibility Issues

- **You will likely want to make your HTML forms accessible**
  - In this case, you should use a `<label>` element that use the for attribute to associate it with the checkbox or radio button

Example

```
<fieldset>
   <legend>Rental Car Options:<legend>
   <input type="checkbox" id="tollTag" name="tollTag"
          value="tolltag" checked>
   <label for="tollTag">Electronic Toll Tag ($3.95/day
          plus tolls)</label><br>
   <input type="checkbox" id="gps" name="gps" value="gls">
   <label for="gps"> GPS ($4.95/day)</label><br>
   <input type="checkbox" id="roadside" name="roadside"
          value="roadside">
   <label for="roadside">Roadside Assistance ($2.95/day)
          </label><br>
</fieldset>

<fieldset>
   <legend>Insurance Option:</legend>
   <input type="checkbox" id="self" name="insurance"
          value="self" checked>
   <label for="self">Self Insured</label><br>
   <input type="checkbox" id="ldw" name="insurance"
          value="ldw">
   <label for="ldw">Loss Damage Waiver</label><br>
</fieldset>
```

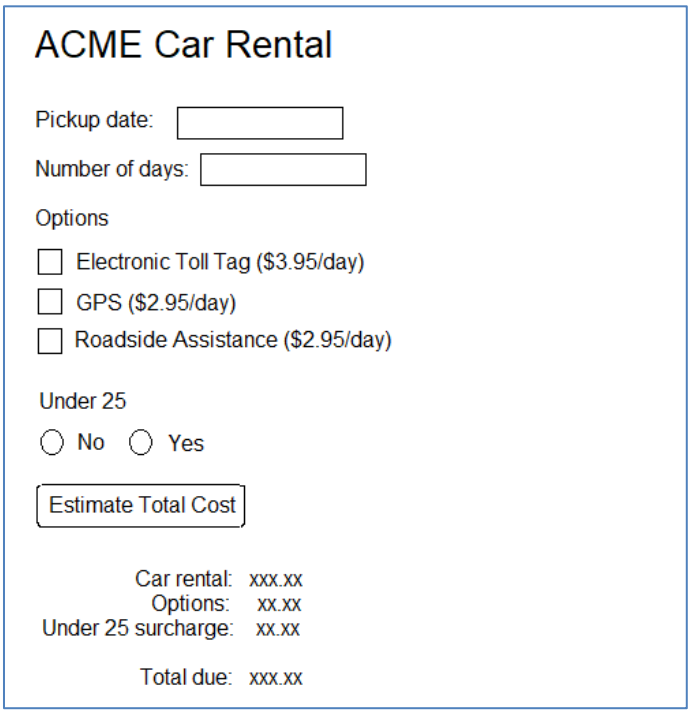
  - NOTE: In the examples in these workbooks, we will not spend much energy in styling the HTML
    - Using the Bootstrap grid system or divs / CSS would make this look better and be more responsive

Exercises

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

EXERCISE 1

Create `index.html` and design it similar to what you see below.



Write code in the `window.onload` event handler to hook up a click event handler for the button. Rules for determining the rental charges are outlined below:

basic car rental is $29.99 per day
there is a 30% surcharge on the basic car rental for drivers under 25

all taxes have already been incorporated into the fees shown

DON'T FORGET TO commit and push!

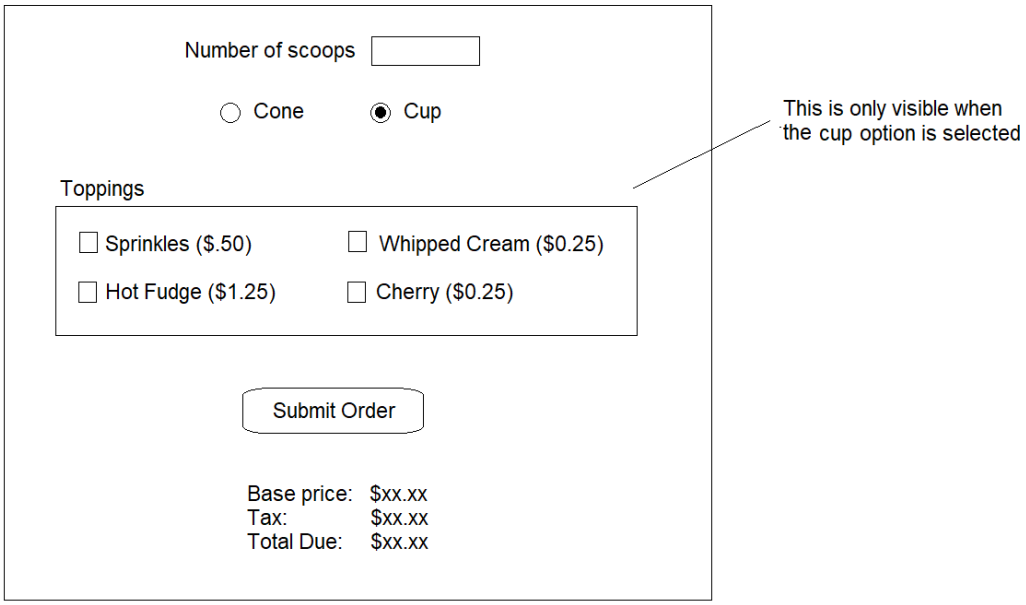
Mini-Project

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

EXERCISE 1

Let's build a web site that allows a user to order an ice cream dessert. Clearly, the site won't submit an order to a worker that will make the ice cream dessert, but it will price out the order.

Your web site should resemble the diagram below, however the behavior of the page is much more important than its appearance.



An ice cream cone or cup with one scoop costs $2.25. Each additional scoop of ice cream is $1.25.

Only cups can have toppings. This means you will need to hide or show the toppings section when the radio buttons for cone and cup are clicked.

Use HTML validation attributes so that the number of scoops is required and must be between 1 and 4.

DON'T FORGET TO commit and push!