

# **Learn to Code**

**JavaScript**

**Student Workbook #2**

Version 4.0 Y

DRAFT

DRAFT

author: Dana L. Wyatt, Ph.D.

# Table of Contents

<b>Module 1 Introduction to JavaScript.....</b>	<b>1-1</b>
Section 1-1 Thinking About Programming.....	1-2
Programming .....	1-3
Section 1-2 Learning to be Precise.....	1-4
Learning to Be Precise .....	1-5
Examples: "Program the Monkey" .....	1-6
Exercises: "Program the Monkey" .....	1-7
Exercises: "Program the Monkey" <i>cont'd.</i> .....	1-8
Expanded Command Vocabulary .....	1-9
Examples: "Smarter Monkey" .....	1-10
Exercises: "Program the Monkey" - Part 2 .....	1-11
Exercises: "Program the Monkey" - Part 2 <i>cont'd.</i> .....	1-12
Section 1-3 Dealing with Ambiguity.....	1-13
Dealing with Ambiguity .....	1-14
Trying to Be Exact .....	1-15
Trying to Be Exact <i>cont'd.</i> .....	1-16
Trying to Be Exact <i>cont'd.</i> .....	1-17
Trying to Be Exact <i>cont'd.</i> .....	1-18
Complex Problems.....	1-19
Exercise.....	1-20
Section 1-4 JavaScript Basics .....	1-21
JavaScript .....	1-22
What Can JavaScript Do? .....	1-23
What Can't JavaScript Do? .....	1-24
JavaScript vs ECMAScript .....	1-25
JavaScript vs ECMAScript <i>cont'd.</i> .....	1-26
JavaScript Code Structure .....	1-27
Using Semicolons to Separate Statements .....	1-28
Declaring Variables.....	1-29
Declaring Variables <i>cont'd.</i> .....	1-30
Storing Values in a Variable.....	1-31
Using <code>console.log()</code> to Display a Message .....	1-32
Using <code>console.log()</code> to Display a Message <i>cont'd.</i> .....	1-33
Comments.....	1-34
Using Strict Mode.....	1-35
Using Strict Mode <i>cont'd.</i> .....	1-36
Primitive Types.....	1-37
Primitive Types <i>cont'd.</i> .....	1-38
Exercises.....	1-39
Getting Visual Studio Code Ready .....	1-40
Installing Extensions.....	1-41
"Code Along" - Hello World: The First Program .....	1-42
"Code Along" - Hello World: The First Program <i>cont'd.</i> .....	1-43
"Code Along" - Hello World: The First Program <i>cont'd.</i> .....	1-44
<b>Module 2 Working with Numbers.....</b>	<b>2-1</b>
Section 2-1 Building Expressions .....	2-2
Expressions .....	2-3
Arithmetic Operators .....	2-4
Arithmetic Operators <i>cont'd.</i> .....	2-5
Examples: Using Arithmetic Operators .....	2-6
Examples: Using Arithmetic Operators <i>cont'd.</i> .....	2-7
Uninitialized Variables .....	2-8
Precedence .....	2-9
Precedence <i>cont'd.</i> .....	2-10

Exercises.....	2-11
Exercises <i>cont'd</i> .....	2-12
Exercises <i>cont'd</i> .....	2-13
Exercises <i>cont'd</i> .....	2-14
The Math Object .....	2-15
The Math Object <i>cont'd</i> .....	2-16
The Math Object <i>cont'd</i> .....	2-17
Rounding Floating Point Numbers.....	2-18
Exercises.....	2-19
Exercises <i>cont'd</i> .....	2-20
Pre/Post Increment/Decrement.....	2-21
Pre/Post Increment/Decrement <i>cont'd</i> .....	2-22
Pre/Post Increment/Decrement <i>cont'd</i> .....	2-23
Assignment Operators.....	2-24
Precedence (again).....	2-25
Section 2-2 Parsing Strings into Numbers.....	2-26
Parsing Strings into Numbers .....	2-27
Using parseInt () .....	2-28
Using parseInt () <i>cont'd</i> .....	2-29
Using parseFloat () .....	2-30
Using Number () .....	2-31
Unary + Also Converts to Number.....	2-32
Exercises.....	2-33
<b>Module 3 Programming with Conditionals.....</b>	<b>3-1</b>
Section 3-1 Making Decisions with an if/else.....	3-2
Making Decisions using the if Statement.....	3-3
Comparison Operators.....	3-4
Using the if Statement.....	3-5
Using the if / else Statement .....	3-6
Exercises.....	3-7
Using an if / else / if Statement.....	3-8
Using an if / else / if Statement <i>cont'd</i> .....	3-9
Exercises.....	3-10
Exercises <i>cont'd</i> .....	3-11
Making And / Or Decisions.....	3-12
Exercises.....	3-13
var vs let .....	3-14
Section 3-2 Making Decisions with a switch.....	3-15
The switch Statement .....	3-16
Using the switch .....	3-17
Using the switch <i>cont'd</i> .....	3-18
Exercises.....	3-19
<b>Module 4 JavaScript in the Browser.....</b>	<b>4-1</b>
Section 4-1 Building HTML Pages that Use JavaScript.....	4-2
The <script> Element.....	4-3
JavaScript Functions .....	4-4
JavaScript Functions <i>cont'd</i> .....	4-5
External Scripts .....	4-6
Organizing Scripts .....	4-7
Section 4-2 Interacting with Page Elements.....	4-8
Accessing Elements on the Page using getElementById.....	4-9
Working with Contents of an Element using innerHTML .....	4-10
Working with <input> Elements.....	4-11
Working with <input> Elements <i>cont'd</i> .....	4-12
Working with <input> Elements <i>cont'd</i> .....	4-13
Combining Finding the HTML Element and Getting the Value .....	4-14

Section 4–3 Event Handling.....	4-15
Events.....	4-16
Event Attributes.....	4-17
Coding Event Logic in an HTML Attribute .....	4-18
Assigning Event Handlers When the Window Finishes Loading.....	4-19
In-Class Demo.....	4-20
Important Note!.....	4-21
Exercises.....	4-22
Exercises <i>cont'd</i> .....	4-23
Exercises <i>cont'd</i> .....	4-24
Adding a Little Error Handling .....	4-25
Adding a Little Error Handling <i>cont'd</i> .....	4-26
Adding a Little Error Handling <i>cont'd</i> .....	4-27
Mini-Project.....	4-28
Appendix A Learning References.....	A-1
Learning References .....	A-2



## **Module 1**

### **Introduction to JavaScript**

Section 1–1

Thinking About Programming

# **Programming**

---

- Programming is essentially the process of telling a computer what to do -- step by step -- in a language the computer understands
- When you program, you have to be exact
  - "Close only counts in horseshoes and hand grenades"
- When you program, you have to use a language the computer understands
  - There are many, maNY, MANY languages out there
  - This week, we will begin our study of JavaScript
- JavaScript is a good language to learn
  - There are over 1.6 billion web sites in the world, and JavaScript is used on 95% of them!

Section 1–2

Learning to be Precise

## Learning to Be Precise

---

- To demonstrate using precise instructions, we'll spend a little time on a paper/pen coding exercise
- In this first phase, we will make a monkey move to his banana and eat it using a set of specific commands
  - Note: the ideas for this exercise came from [codemonkey.com](http://codemonkey.com)
- Right now, your monkey understands 3 commands

Step *number*

where *number* is the number of squares to move

Example: Step 5

Turn *direction*

where *direction* is left or right

Example: Turn left

Eat banana

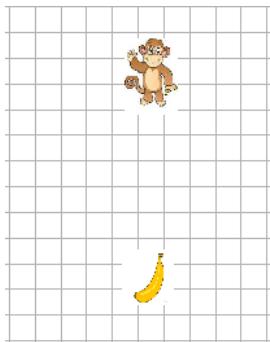
This only works if the monkey is standing on an adjacent square to the banana and facing it.

Example: Eat banana

## Examples: "Program the Monkey"

### Example

Move the monkey to the banana and have him eat it.

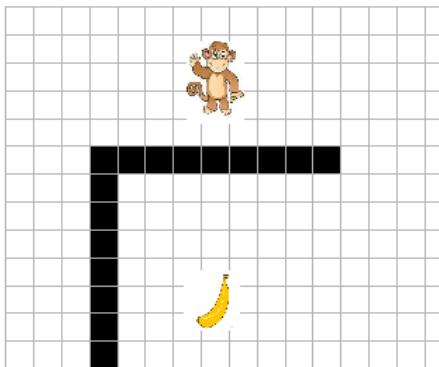


#### SOLUTION

Step 6  
Eat banana

### Example

Move the monkey to the banana and have him eat it. Avoid the barrier. Pay attention to the direction the monkey is facing.



#### SOLUTION

Turn left  
Step 5  
Turn right  
Step 7  
Turn right  
Step 4  
Eat banana

- Note: Just like programming in real life, there is often more than one way to be successful

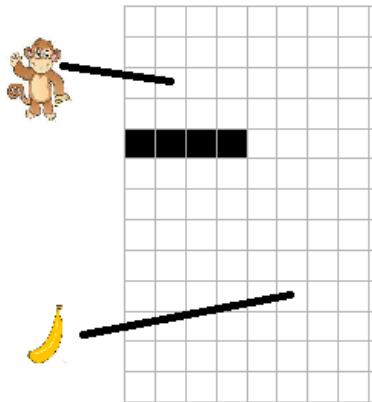
## **Exercises: "Program the Monkey"**

---

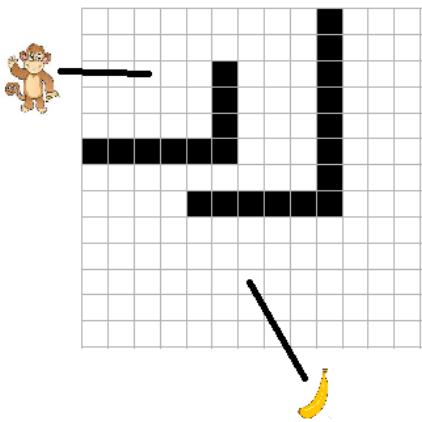
In the next few exercises, we will let you figure out the steps needed to "program your monkey" to move to and eat the banana. We are using lines to show you where the monkey and banana are so that the positions are very precise.

When you finish, talk it over in your group to see how others programmed their monkey.

### EXERCISE 1

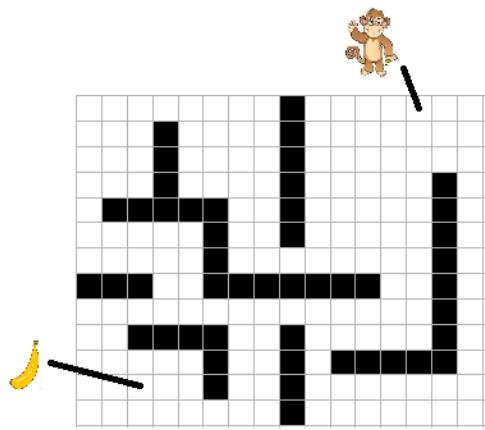


### EXERCISE 2



## **Exercises: "Program the Monkey" cont'd**

## EXERCISE 3



## Expanded Command Vocabulary

- When you are learning a programming language, you start with just a few commands and learn to use them
  - When you conquer them, you learn more commands and how to use them too!
  - Before long, you know a LOT about the programming language
- Your monkey understands the commands we discussed before, plus some commands that can be used to pick up an item or drop an item

:

Step *number*  
where *number* is the number of squares to move

Turn *direction*  
where *direction* is left or right

Pickup *item*  
where *item* is banana or basket

When you pickup an item, you must be facing the item in an adjacent square.

Example: Pickup basket

Drop *item*  
where *item* is banana or basket

When you drop an item, it stays in the square directly in front of the square you are standing in. NOTE: Only one item can reside on a square, however, if the banana is IN the basket that counts as one item.

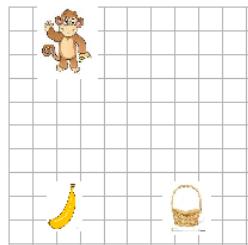
Example: Drop basket

Eat banana  
This only works if the monkey is standing on square adjacent to the banana directly facing it.

## Examples: "Smarter Monkey"

### Example

Make the monkey drop the banana into the basket.



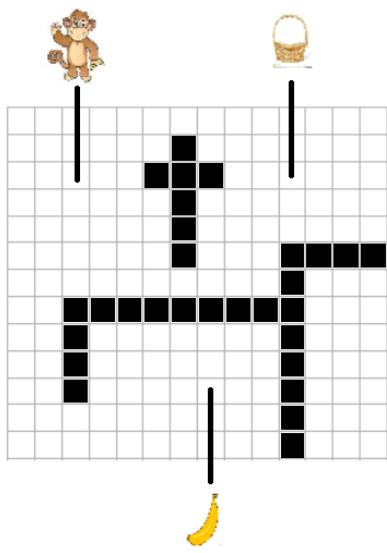
#### SOLUTION

Step 5  
Pickup banana  
Turn left  
Step 5  
Turn right  
Drop banana

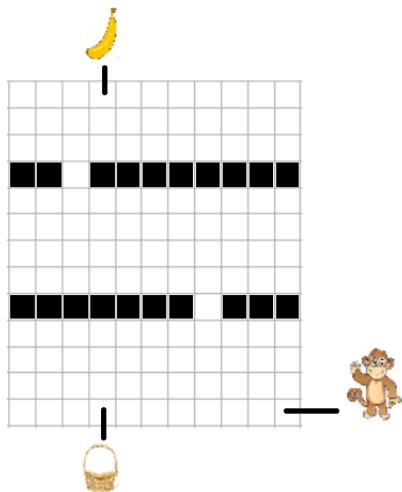
## Exercises: "Program the Monkey" - Part 2

In these exercises, you need to "program your monkey" to put the banana in the basket. Like before, when you finish, talk it over in your group to see how others programmed their monkey.

### EXERCISE 1



### EXERCISE 2



## **Exercises: "Program the Monkey" - Part 2**

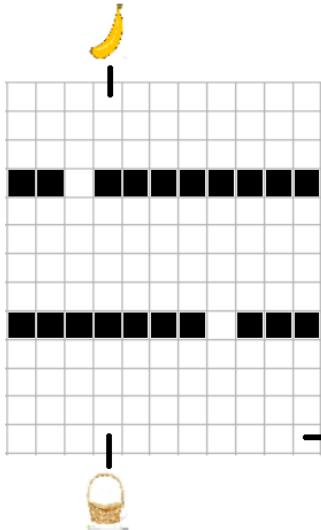
*cont'd*

---

### EXERCISE 3

Would your answer change if you knew that the goal was to take the smallest number of steps possible?

The goal here is: *the banana ends up in the basket*. It doesn't say where the basket has to be. Do you pick up the banana and carry it to the basket? Do you take the basket over near the banana?



Section 1–3

Dealing with Ambiguity

## **Dealing with Ambiguity**

---

- Sometimes, it's harder to generate a list of the tasks you must perform when the process isn't quite as well defined or as visual as working with monkeys!
- In these situations, we often write down the steps that we need to perform in English in order to "get our head around" the problem
- Computers follow your exact instructions -- but learning to be precise is tricky!
- Watch the "Exact Instructions Challenge" in You Tube before proceeding any further!

<https://www.youtube.com/watch?v=Ct-1OOUqmY>

## Trying to Be Exact

---

- How do you make scrambled eggs for two people?

### ATTEMPT 1

Get four eggs from refrigerator  
Crack eggs into skillet  
Scramble eggs

- Do you understand the process from our first attempt
- What's wrong with the description above?

- The write up has some pre-conditions to scrambling eggs, but it doesn't really tell us how to scramble eggs

### ATTEMPT 2

Get four eggs from refrigerator  
Crack eggs into skillet

Repeat  
[  
    Stir eggs as they cook  
    Wait 30 seconds  
] until they are the desired consistency

Turn stove off

- Better, but where did the eggs come from?

## Trying to Be Exact *cont'd*

---

- How exact do you have to be?

- A lot of that depends on how intelligent the reader is

### ATTEMPT 3

```
Open the refrigerator  
Get four eggs  
Close refrigerator
```

```
Crack eggs into skillet
```

```
Repeat  
[  
    Stir eggs as they cook  
    Wait 30 seconds  
] until they are the desired consistency
```

```
Turn stove off
```

- Of course, if we are programming a computer - it has no preconceived insights and you have to be very exact

## Trying to Be Exact *cont'd*

---

- You also have to describe possible error conditions

- For example, what if we are out of eggs?

### ATTEMPT 4

```
Open the refrigerator
If we are out of eggs
[
    Make different breakfast plans
    Close refrigerator
    Exit script
]
Get four eggs
Close refrigerator

Crack eggs into skillet

Repeat
[
    Stir eggs as they cook
    Wait 30 seconds
] until they are the desired consistency

Turn stove off
```

## Trying to Be Exact *cont'd*

---

- Not bad -- but are you really being detailed?

- Where did the skillet come from?
  - Will your eggs might stick to the pan as you cook?

### ATTEMPT 5

```
Open the refrigerator
If we are out of eggs
[
    Make different breakfast plans
    Close refrigerator
    Exit script
]
Get four eggs
Close refrigerator
```

```
Get skillet from rack above stove
Spray PAM all around skillet for 3 seconds
Put skillet on stove
```

```
Crack eggs into skillet
```

```
Repeat
[
    Stir eggs as they cook
    Wait 30 seconds
] until they are the desired consistency
```

```
Turn stove off
```

- Can you see your sarcastic friend following the instructions for spraying PAM?

- My sarcastic friend would spray the handle and bottom of the skillet!
  - Oops! Replace with:

```
Spray PAM all around the inside of the skillet for 3 seconds
```

## Complex Problems

---

- **When you are working with complex problems, you really have to think about all aspects of the problem**

- Where did the PAM come from?
- Do they know how to crack eggs?
- What if some of the shell ended up in the pan?
- What did they do with the shells once they put the egg in the skillet?
- Who turned on the stove?
- Stir the eggs with what?
- What is "desired consistency"?
  - \* In my house, one of us likes eggs runny and the rest want them cooked into dried hard unappetizing flecks!"
- Is plating the eggs part of the scope of the problem?
- Is cleaning up your mess part of the scope of the problem?

- **Other things that might impact how you write include:**

- The vocabulary of the user (new to the process, experienced)
- The skills of the user (never cooked, casual cook, chef)
- The complexity of the process
  - \* Sometimes, you just won't understand enough about the process to get all of the details right at the beginning
  - \* Software development is an iterative process!

## **Exercise**

---

### **EXERCISE 1**

Write out the process of how to "brush your teeth". This may take 4-10 minutes if you do a good job.

Then, meet with your group to go over the write-ups. Tweak the write-ups as needed.

Your team will "present" a "brush your teeth" process to the class as a whole. Your team can select one of the team member's to advance or can blend them together using steps from each team members.

Section 1–4

JavaScript Basics

# JavaScript

---

- **JavaScript was initially created to provide dynamic behavior in web pages**
  - Programs in this language are called scripts and they are written between the `<script>` tags in a web page's HTML
  - This is often called *client-side scripting*
- **JavaScript is fully integrated with HTML/CSS**
  - It is supported by all modern browser
- **Today, JavaScript can also execute on a server**
  - This allows it to provide back-end processing
  - This is often called *server-side scripting*
- **In fact, today JavaScript can execute on any device that the JavaScript engine**
  - Mobile applications can even be developed using JavaScript
- **Unlike many other languages, JavaScript scripts are executed as plain text and don't need compilers to translate them into machine code first**

## What Can JavaScript Do?

---

- **Most of the scripts you write as a Web Developer will be client-side scripts**
  - You will include them in your HTML pages
- **Their primary purpose will be to make things happen in the browser**
  - Hide and show form fields
  - Validate form data
  - Submit data to a server for processing
  - Fetch data from a server asynchronously
  - Dynamically create portions of the page
- **When JavaScript runs on a server, JavaScript can do other types of things, including:**
  - Read/write files
  - Interact with databases
  - Perform network requests

## **What Can't JavaScript Do?**

---

- **When JavaScript executes in the browser, its abilities are somewhat limited**
  - This is to keep a malicious web page from accessing the user's private information or corrupting the user's data
- **JavaScript on a webpage may not:**
  - interact with the camera/mic without a user's explicit permission
  - read/write most files on the hard disk (with some exceptions)
  - directly access operating system functions

## **JavaScript vs ECMAScript**

---

- **In the early days of the web, browser manufacturers each used their own proprietary language**
  - But this meant web developers had to pick a browser and code just for it
  - This was not ideal!
- **In 1995, JavaScript was created for Netscape**
  - It was eventually adopted by Mozilla for Firefox and became an ECMA standard in 1997
  - Standardization is important so that web developers can write one set of code and know that it will run in all browsers that conform to that standard
- **ECMAScript is the name of the language standard**
  - JavaScript is the name of the language that implements the standard
- **Early versions of the standard were:**
  - ECMAScript 1 (1997)
  - ...
  - ECMAScript 5 (2009)
  - ECMAScript 5.1 (2011)
- **These are often called ES1 through ES5**

## JavaScript vs ECMAScript *cont'd*

---

- Beginning in 2015, ECMAScript is referred to by either the year or by the ES name
  - ECMAScript 2015 (or ES6)
  - ECMAScript 2016 (or ES7)
  - ECMAScript 2017 (or ES8)
  - ECMAScript 2018 (or ES9)
- With all these changes, browsers are constantly playing catch up
  - You can be sure that *all browsers* support ECMAScript 3
  - *All modern browsers* fully support ECMAScript 5
  - Go to [https://www.w3schools.com/js/js\\_versions.asp](https://www.w3schools.com/js/js_versions.asp) to see which version of ECMAScript each browser supports

## JavaScript Code Structure

---

- A JavaScript script contains one or more lines of JavaScript code
  - Each statement is typically written on a separate line
  - Each statement usually ends with a semicolon ( ; )
- JavaScript ignores whitespace unless it is part of a quoted string

### Example

```
var message = "Welcome to JavaScript";
var name      =      "Natalie";
var age=37;

var outputString = "Name: " + name +
                  " Age: " + age;
```

- JavaScript scripts are either contained within the <script> tags in an HTML page or stored in a file that ends with .js
  - For example: process\_order.js

# Using Semicolons to Separate Statements

---

- Each statement typically ends with a semicolon (;)
  - If statements appear next to each other on the same line, the semicolon is *required*

## Example

```
// semicolon required
var name = "Natalie"; var age = 37;
```

- If the statement ends with a newline, the semicolon is *optional*

## Example

```
// semicolon not required
var name = "Natalie"
var age = 37
```

- Best practice says to always use the semicolon
- However, semicolons should not be included after block statements
  - Block statements use curly brackets { } to define the beginning and end of the block

## Example

```
var age = 12;
var price = 19.95;

if (age >= 65) {
    price = price * .9;
} // -----no semicolon here
```

## Declaring Variables

---

- When you write JavaScript to operate on data, you will use *variables* to hold your data
  - A variable might hold a name you extract from an input box in an HTML form
  - A variable might reference a div where you are going to place a message
- You create a variable by putting the keyword **var** in front of it

### Example

```
var name;  
var age;
```

- Variable names can be made up of numbers, letters, and the characters \$ or \_
  - They are case sensitive
  - They may not contain spaces or start with a number

### Example

```
var NAME;  
var Name;  
var name; // all 3 are allowed because of case sensitivity  
          // but that doesn't mean you should do this!
```

### Example

```
var 1stPlaceWinner; // invalid because it starts with a number
```

## Declaring Variables *cont'd*

---

- You can declare more than one variable on a line by separating them with commas

### Example

```
var name, age;
```

- You can't use any of JavaScript's keywords as a variable name
  - Keywords are words that have a predefined behavior in JavaScript
    - \* For example, var, if, and function
  - Using these would cause the JavaScript parser to become confused
- Best practice says:
  - Start variables with a lower case letter
  - Capitalize the first letter of each additional word in the variable name (this is known as camelCase)

### Example

```
var firstName;  
var lastName;  
var nameOfUniversity;
```

## Storing Values in a Variable

---

- In JavaScript, you can store a value in a variable using the assignment operator

- If no initial value is specified, it holds the special value `undefined`

### Example

```
var name;  
name = "Natalie";
```

- Variables can store any JavaScript data type, including strings or numbers

### Example

```
var name, age;  
  
name = "Natalie";      // strings are surrounded by quotes  
age = 37;
```

- Assignment always goes from right to left

- The value to the right of the assignment operator is computed before the assignment occurs

### Example

```
var price, discount;  
  
price = 22.50;  
  
// the discount is 10% of the price  
discount = price * .1;    // multiply price by .1 first  
                          // and then do the assignment
```

## Using `console.log()` to Display a Message

---

- JavaScript can "display" data in different ways
    - We will start with `console.log()`
  - The `console.log()` function writes text to a console window
- Example**
- ```
console.log("Hello world");
```
- When running under Node.js, the output will appear in an output window in Visual Studio Code
  - When running in a browser, the output will appear in the Developer's Tool window
    - \* In many browsers, it can be displayed by pressing F12
- `console.log()` can also write the value of variables

**Example**

- Often, you concatenate your data to the end of a string using a plus (+) to provide a "label" to your message

## Using `console.log()` to Display a Message *cont'd*

---

### Example

```
var name;  
name = "Zachary";  
console.log("Your name is " + name);
```

## Comments

---

- **JavaScript comments are used to leave notes in code to you or other programmers about how the code works**
- **There are two ways to write comments**

- You can use `//` to tell the JavaScript parser to ignore the rest of the text on the current line

### Example

```
var price, discount;  
  
price = 22.50;  
  
// find the discount amount if the discount is 10%  
discount = price * .1;      // multiply price by .1 first  
                           // and then do the assignment
```

- You can create a multi-line comment by starting with `/*` and ending with `*/`

### Example

```
var price, discount;  
  
price = 22.50;  
  
// find the discount amount if the discount is 10%  
discount = price * .1;      /* multiply price by .1 first  
                           and then do the assignment */
```

- **JavaScript comments are visible to someone viewing the source code in a browser's Developer Tools**
- So don't write something you don't want seen by smart users

## Using Strict Mode

---

- Although we have shown how to define variables, technically the language does not require it
- If a variable is not explicitly declared, JavaScript implicitly (automatically) declares it

### Example

```
name = "Natalie"      // assigns name a value
age = 36;             // assigns age a value
```

- The problem with this is that a misspelled variable can cause you lots of headaches
  - This is because the JavaScript engine just creates the misspelled one on the fly

### Example

```
var number;
number = 10;
...
numbers = 12;
...
console.log(number);    // you are expecting 12 but
                        // it displays 10
```

- ECMAScript 5 introduced a "strict mode" to JavaScript that dis-allows undeclared variables
  - It is ignored by earlier versions of JavaScript

## Using Strict Mode *cont'd*

---

- To use strict mode, you can place "use strict" at the top of a JavaScript file

### Example

```
"use strict";

var number;
number = 10;
...
numbers = 12;           // Error; numbers is undefined
...
console.log(number);
```

- Although we may not write "use strict" at the top of all our examples, you can assume it is there!

- Strict mode is supported in:

- Chrome 13+
- Internet Explorer 10+ (recently deprecated)
- Edge 12+
- Firefox 4+
- Safari 5.1+
- Opera 12+

## Primitive Types

---

- JavaScript provides seven different data types, two of which are listed below
  - Number
  - String
- NOTE: we will learn about others later

### Example

```
"use strict";  
  
var word, num;  
  
word = "banana";           // a String  
num = 12;                 // a Number
```

- You can perform mathematical operations on a number, but not on a string

### Example

```
"use strict";  
  
var word, num;  
  
word = "banana";  
num = 12;  
  
num = num + 1;           // num is now 13  
  
word = word + 1;         // error! can't add 1  
                        // to "banana"
```

## Primitive Types *cont'd*

---

- But you can concatenate two strings using the + operator

### Example

```
"use strict";  
  
var phrase1, phrase2, wholePhrase;  
  
phrase1= "Learning to program ";  
phrase2= "is fun!";  
  
wholePhrase = phrase1 + phrase2;  
    // wholePhrase contains "Learning to program is fun!"
```

# Exercises

---

## EXERCISE 1

Use `var` to create variables with good names for the following:

- a customer id
- a customer's first, middle, and last name
- their gender
- their date of birth
- their driver's license number
- their auto policy number

Example:

```
var customerId;
```

## EXERCISE 2

Go to [https://www.w3schools.com/js/js\\_reserved.asp](https://www.w3schools.com/js/js_reserved.asp) and answer the following questions.

1. What are 10 of the reserved words that you can't use for variable names?
2. What are the names of 10 of the JavaScript built-in objects, properties, and methods that you should not use for variable names?
3. What are 10 of the HTML and Window objects names that you should not use for variable names?
4. What are 6 of HTML event names that you should not use for variable names:

## EXERCISE 3

Write JavaScript to declare variables for your name, as well as the city and state you were born in. Then assign those variables values for represent you!

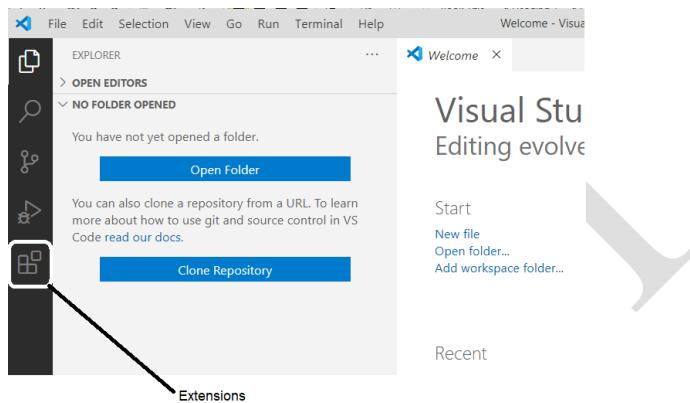
## Getting Visual Studio Code Ready

---

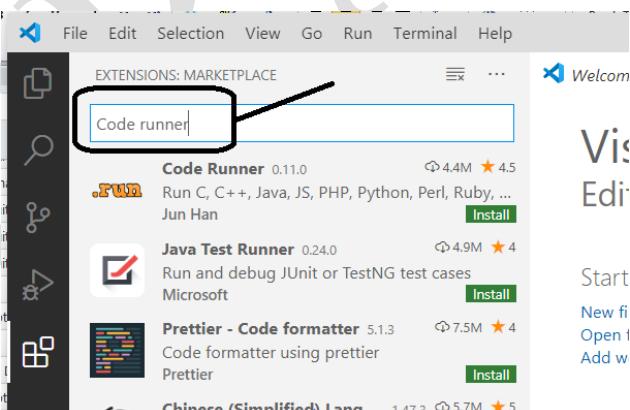
- You used Visual Studio Code in the HTML/CSS/Bootstrap portion of the class, so you should be familiar
- Before we begin coding, we are going to add a couple of extensions
  - **Code Runner** allows us to run JavaScript scripts without having to embed them in an HTML page
  - **Live Server** creates a local development server that hosts the pages open in Visual Studio Code
    - \* This keeps us from having to deploy a site to a web server to test it

# Installing Extensions

- To install extensions, click on the Extensions icon on the left bar



- In the search pane, type the name of the extension you want to install



- Then click the **Install** button
- You may have to close and then re-open Visual Studio Code in order to see your extensions

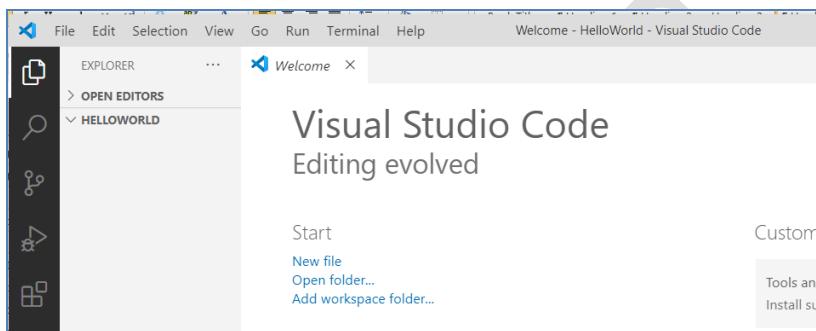
## **"Code Along" - Hello World: The First Program**

---

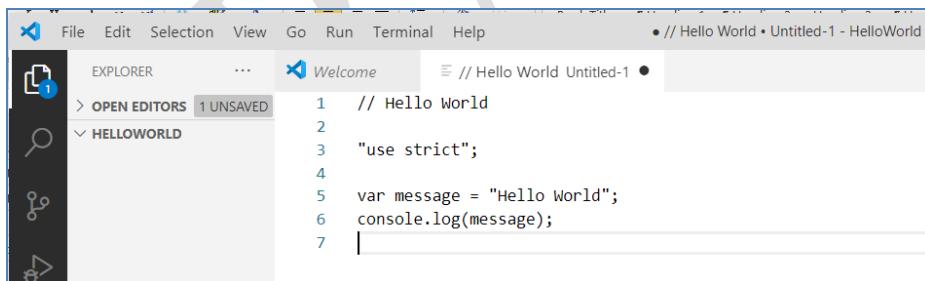
- Everybody's first program is "Hello World"
  - It simply writes a message to the screen
- We will do that now...
- To get started, create a Git repository called **LearnToCode**
  - You can put it in your working directory
  - We will work in this same repo for all of our JavaScript exercises
- In the repo, add a subfolder named **Workbook2\_Mod01**
  - As we go through the JavaScript portion of the course, we will place our scripts in a folder whose name matches the workbook and module
- Finally, in the **Workbook2\_Mod01** folder, add a subfolder named **HelloWorld**
  - Each exercise will go in its own subfolder

## "Code Along" - Hello World: The First Program cont'd

- In Visual Studio Code, use the **File > Open Folder** menu to navigate to your **Workbook2\Mod01\HelloWorld** folder



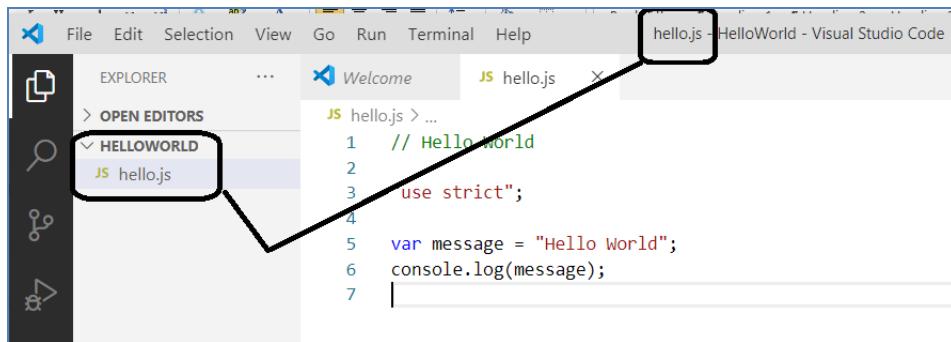
- Use the **File > New File** menu to create a text file, then type the code you see below



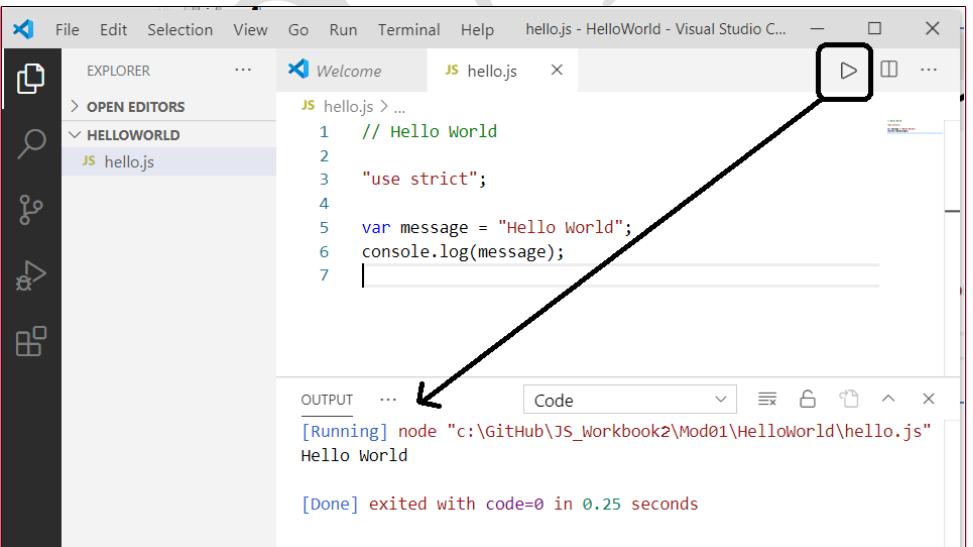
- You can see on the far right of the menu bar, the file is named Untitled-1
- To save this file, use **File > Save As** and name it **hello.js**

## "Code Along" - Hello World: The First Program cont'd

- The file will now show up in the Explorer window



- To run the script, click the run icon in the top right corner and then look at the Output pane for the results



Commented [Ma1]: Replace this image w/ correct path

**Module 2**

**Working with Numbers**

Section 2–1

Building Expressions

# Expressions

---

- A JavaScript expression is any set of variables, literals, and operators that evaluate to a single value

## Example

```
// assuming num1 and num2 are the names of variables  
  
42  
num1  
num1 + 1  
(num1 + num2) / 2
```

- Expressions are often assigned to a variable

## Example

```
num1 = 42;  
num2 = num1;  
nextValue = num1 + 1;  
average = (num1 + num2) / 2;
```

- Over the next few pages, we will examine the arithmetic operators available to JavaScript programmers
  - Over the next few weeks, we will examine other operators as well including comparison, logical, string, and more

## Arithmetic Operators

---

- JavaScript has both unary and binary arithmetic operators that you can use to code complex formulas
  - An operator is binary if it has two operands
  - An operator is unary if it has a single operand
- The binary operators include:
  - Addition ( + )
  - Subtraction ( - )
  - Multiplication ( \* )
  - Division ( / )
  - Modulo (or division remainder) ( % )

### Example

```
var num1 = 15, num2 = 10;
var answer;

answer = num1 + num2;          // answer is 25
answer = num1 - num2;          // answer is 5

answer = num1 * num2;          // answer is 150
answer = num1 / num2;          // answer is 1.5

answer = num1 % num2;          // answer is 5
```

## Arithmetic Operators *cont'd*

---

- The unary operator **negate ( - )** allows you to reverse the sign on a number

### Example

```
var num1 = 15, num2 = -10;  
var answer;  
  
answer = -num1;           // answer is -15  
answer = -num2;           // answer is 10
```

- ES7 introduced an operator for exponentiation ( **\*\*** )

### Example

```
var num1 = 15;  
var answer;  
  
// if supported by your browser  
answer = num1 ** 2;        // answer is 225
```

- If your browser doesn't support exponentiations, we will learn another way to raise a number to a power shortly
  - You can use `Math.pow()`
- The operators themselves are fairly easy to understand
  - The trick as a programmer is to find a way to use them to solve problems
- We will spend quite a bit of time this first week trying to learn to "think like a programmer"!

## Examples: Using Arithmetic Operators

---

- Addition ( + )

### Example

```
var costOfVetVisit = 70.0;  
var vaccinationFee = 37.50;  
var totalDue = costOfVetVisit + vaccinationFee;
```

- Subtraction ( - )

### Example

```
var priceOfMeal = 72.85;  
var discountCoupon = 10;  
var totalDue = priceOfMeal - discountCoupon;
```

- Multiplication ( \* )

### Example

```
var pricePerGallon = 2.97;  
var numberOfGallons = 11.3;  
var totalDue = pricePerGallon * numberOfGallons;
```

- Division ( / )

### Example

```
var costOfLimo = 300;  
var numberOfPromGoers = 2;  
var costPerPerson = costOfLimo / numberOfPromGoers;
```

## Examples: Using Arithmetic Operators

*cont'd*

---

- Modulo ( % )

### Example

```
// Do something if a number is even  
  
var num = 63;  
if (num % 2 == 0) {  
    ...  
}
```

- Exponentiation ( \*\* )

### Example

```
var sideOfSquare = 5;  
var areaOfSquare = sideOfSquare ** 2;
```

## Uninitialized Variables

---

- When JavaScript variables are declared, they have an initial value of undefined

### Example

```
var num;  
// at this point, num has the value 'undefined'  
  
var word;  
// at this point, word has the value 'undefined'
```

- Because variables in JavaScript don't have a data type (like string or number), the name doesn't affect any initial value
  - \* If you've ever taken a C# or Java class (or many other languages), those variables are explicitly given a type (ex: int num; string word;) and the initial value matches the type (0 for integers and "" for strings)

- If you do any mathematical operation or string concatenation on an undefined variable, the result will be NaN

- NaN means "Not a Number"

### Example

```
var num;  
num = num + 1;  
// at this point, num has the value 'NaN'
```

- Important: make sure your variables have values before you try and use them

## Precedence

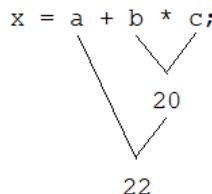
---

- Operator precedence describes the order in which operations are performed in an expression
  - It is similar to the algebraic precedence most of you learned in your early math classes

### Example

```
var a = 2;  
var b = 5;  
var c = 4;  
  
var x = a + b * c;           // result is 22
```

- How did we come up with answer? Multiplication and division have higher precedence than addition and subtraction



- Precedence can be changed by using parentheses

### Example

```
var a = 2;  
var b = 5;  
var c = 4;  
  
var x = (a + b) * c;           // result is 28
```

## Precedence *cont'd*

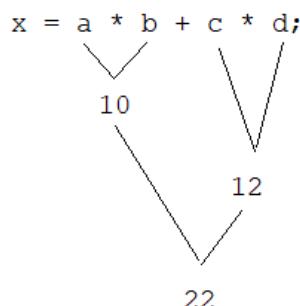
---

- When operations have the same precedence, they are performed left to right

### Example

```
var a = 2;  
var b = 5;  
var c = 4;  
var d = 3;  
  
var x = a * b + c * d;           // result is 22
```

- How did we calculate that result?



# Exercises

---

## EXERCISE 1

In this exercise, you will use Visual Studio Code to create and then run a script.

SAMPLE PROBLEM: How do you calculate the total due at a restaurant given the food cost, the tax and the tip?

FORMULA: Total Due is determined by: Food Cost + Tax + Tip

SCRIPT:

```
// sample inputs
var foodCost = 79.25;
var tax = 6.54;
var tip = 12.00;

// calculation
var totalDue = foodCost + tax + tip;

// output
console.log("The total due is " + totalDue);
```

To create the script, follow the instructions below:

1. Create a new folder in your repo named `Workbook2_Mod02`
2. Create a subfolder in `Workbook2_Mod02` named `Demo`. This is where you will put all of the code for this "project".
3. Launch Visual Studio Code.
4. Use the `File > Open Folder` menu to navigate to and you're your `Demo` folder.
5. Use the `File > New File` menu to create a code file named `demo.js`.
6. Now type the following code into the file:

```
// sample inputs
var foodCost = 79.25;
var tax = 6.54;
var tip = 12.00;
```

## Exercises *cont'd*

---

```
// calculations
var totalDue = foodCost + tax + tip;

// output
console.log("The total due is " + totalDue);
```

7. Use the **File > Save** menu to save your script.
8. There are two ways to run your code. You can click the run icon like you did in the last module or you can use the **Run > Start Debugging** menu to run your script. You will see the output in a window at the bottom.



A screenshot of a terminal window. At the top, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The DEBUG CONSOLE tab is active. The window shows the command `C:\Program Files (x86)\nodejs\node.exe --inspect-brk=39740 demo.js` being run. The output shows the debugger listening on port 39740 and the total due calculated as 97.79. The file name `demo.js:5` is shown at the bottom right.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
C:\Program Files (x86)\nodejs\node.exe --inspect-brk=39740 demo.js
Debugger listening on ws://127.0.0.1:39740/96a547ef-6ae6-4473-8ca9-02374c989f
35
The total due is 97.79
demo.js:5
```

9. Now change the `console.log()` statement to the following to see how to build more complicated output.

```
console.log(
  "Food cost is " + foodCost + " and tax is " + tax);
console.log("Tip is " + tip);
console.log("Total Due is " + totalDue);
```

10. Use **File > Save** to save your script changes and then re-run your script.

Do you see how the tip amount doesn't have two digits to the right of the decimal point? Replace the line that displayed the tip amount with the following.

```
console.log("Tip is " + tip.toFixed(2));
```

Save your script and re-run it. Does it look better?

11. When you are finished, use the **File > Close Folder** menu to close the project. You will now be ready to repeat this process for the next exercise.

## **Exercises** cont'd

### **EXERCISE 2**

Create a subfolder in Mod02 named SimpleMathScripts.

For this exercise, gather in small groups to find formulas and code scripts.

Follow the process outlined in the last exercise to create and run a script. You will create one code file for each problem below. When you run a script, it will run the script that is in the currently open editor window.

1. How do you calculate your net worth given your assets and debts? Figure out the formula and what the script would look like.

Now, code your script in a file named **net\_worth.js**

Your output should be formatted as follows:

Your net worth is *some-number*

When your team gets the script working, move on.

2. How do you calculate the area of a rectangle? Figure out the formula and what the script would look like.

Now, code your script in a file named **area\_of\_rectangle.js**

Your output should be formatted as follows:

The area of the rectangle is *some-number*

When your team gets the script working, move on.

3. How do you calculate the tip amount of a food bill given the tip percentage?

Now, code your script in a file named **tip\_amount.js**

Your output should be formatted as follows:

The tip on a \$*some-number* food bill is \$*some-number*

## Exercises *cont'd*

---

When your team gets the script working, move on.

4. How do you calculate the area of a circle?

Now, code your script in a file named **area\_of\_circle.js**

Your output should be formatted as follows:

The area of a circle with radius *some-number* is *some-number*

When your team gets the script working, move on.

5. How long will it take a savings account worth X to double in value given an interest rate of IR? (Hint: look up the rule of 72)

Now, code your script in a file named **rule\_of\_72.js**

Your output should be formatted as follows:

At a *some-number*% interest rate, your savings account will be worth *some-number* in *some-number* years

Show your doubled balance with 2 digits to the right of the decimal point by calling `.toFixed(2)`. Also show your years with 1 digit to the right of the decimal point using a similar technique.

Where you able to get all of your scripts to work?

Commit your changes. Then use the File > Close Folder menu to close this "project".

## The Math Object

---

- The JavaScript Math object allows you to perform mathematical tasks
- It contains a property named PI
  - `Math.PI` returns PI (approx. 3.14159)

### Example

```
var radius = 5;  
var areaOfCircle = Math.PI * (radius ** 2);
```

- It has algebraic functions that feel familiar, including:
  - `Math.abs(x)` returns the absolute value of x
  - `Math.pow(x, y)` returns the value of x to the power of y
  - `Math.sqrt(x)` returns the square root of x

### Example

```
var originalPrice = 9.97;  
var newPrice = 12.50;  
var difference = Math.abs(originalPrice - newPrice);
```

### Example

```
var radius = 5;  
var areaOfCircle = Math.PI * Math.pow(radius, 2);
```

### Example

```
var num = 5;  
var sqrtOfNum = Math.sqrt(num);
```

## The Math Object cont'd

---

- It has functions that will return the round or truncate numbers, including:
  - `Math.ceil(x)` returns x rounded up to the nearest integer
    - \* With `ceil`, 7.8 becomes 8 and -7.8 becomes -7
  - `Math.floor(x)` returns x rounded down to the nearest integer
    - \* With `floor`, 7.8 becomes 7 and -7.8 becomes -8
  - `Math.round(x)` rounds a number to the nearest integer
    - \* With `round`, 7.8 becomes 8 and -7.8 becomes -8
  - `Math.trunc(x)` returns the integer part of a number
    - \* With `trunc`, 7.8 becomes 7 and -7.8 becomes -7

### Example

```
var numPeople = 118;  
  
var numDozenDonuts = numPeople / 12;    // returns 9.83333  
  
numDozenDonuts = Math.ceil(numDozenDonuts);    // returns 10
```

### Example

```
var billTotal = 82.47;  
var tipAmount = billTotal * 0.2;  
  
var totalToPay = billAmount + tipAmount;    // returns 98.964  
  
// pay with dollar bills (no change); cheapskate  
totalToPay = Math.floor(totalToPay);    // returns 98
```

## The Math Object cont'd

- It has functions that will return the smaller or largest of a set of numbers:

- `Math.min(x, y, ..., n)` returns the number with the lowest value
- `Math.max(x, y, ..., n)` returns the number with the highest value

### Example

```
var test1 = 92, test2 = 98, test3 = 82;  
  
var lowestTestScore = Math.min(test1, test2, test3);  
var highestTestScore = Math.max(test1, test2, test3);
```

- It has a function that returns a random number:

- `Math.random()` returns a random number between 0 and 1

### Example

```
var randNum = Math.random();  
  
// scale this to a number of seconds between 10-15  
var numSec = 10000 + (randNum * 5000);  
  
// now schedule an ad to change after numSec time period
```

- It also has many trigonometric functions, including

- `Math.sin(x)` returns the sine x
- `Math.cos(x)` returns the cosine x

## Rounding Floating Point Numbers

---

- The `Math.round()` function returns the value of a number rounded to the nearest integer

### Example

```
var num1 = 18.49, num2 = 18.5;  
  
var x = Math.round(num1);      // 18  
var y = Math.round(num2);      // 19
```

- Rounding numbers with decimal precision requires a little calculation (i.e. 2 digits to the right of the decimal point)

- Here are two techniques

### Example

```
var price = 7.99;  
var taxRate = .0825;  
  
var tax = price * taxRate;    // returns .659175  
  
// Multiply by 100 and then round to an integer to lose the  
// extra digits. Then divide by 100 to move the decimal point  
  
tax = Math.round(tax * 100) / 100;    // returns .66
```

### Example

```
var price = 7.99;  
var taxRate = .0825;  
var tax = price * taxRate;    // returns .659175  
  
// The toFixed(2) method call returns the value as a  
// string with 2 digits to the right of the decimal  
// point. parseFloat() turns the string back to a number  
  
tax = parseFloat(tax.toFixed(2));    // returns .66
```

## Exercises

---

Create a subfolder in Mod02 named MoreMathScripts. These exercises should be placed there.

### EXERCISE 1

These problems will be a little harder. Gather in groups of 2 or 3 to find work together. However, after you figure out the formulas, try to code the scripts individually.

1. How do you convert a Fahrenheit temperature into Celsius?

Code the script in a file named **f\_to\_c.js**

2. How do you convert a Celsius temperature into Fahrenheit?

Code the script in a file named **c\_to\_f.js**

Now wait for all of your team members to get finished. Review each other's code. Were they similar? The smaller a program is, the more likely the code is to be very similar.

Now, continue the process:

3. Federal taxes are 23% of your salary every month. You make X amount of money. How much is withheld for taxes?

Code the script in a file named **taxes.js**

Make sure not to display partial pennies. Display taxes with 2 digits of precision using `toFixed()`.

4. How do you find the distance between (x1,y1) and (x2,y2)?

Code the calculate distance in a script named **distance.js**

## Exercises *cont'd*

---

Now wait for all of your team members to get finished. Review each other's code. If anyone is struggling, discuss how you solved the problem with them.

Now, continue one more time:

5. You have a room whose dimensions are length x width feet. You are going to tile the room and there are 12 1 foot x 1 foot tiles per box. How many boxes of tiles do you need? You cannot buy a partial box.

You want to buy 10% more tiles than you actually need to handle chips, breakage, and mess-ups. How many boxes will you buy?

Code the script in a file named **tiles.js**

6. There are X people going on a tour. Charter vans seat 15 passengers each. How many vans do you need? Vans cost \$250 day to rent (including cost of the driver). How much will it cost to rent the vans? What is that cost if you split it per person?

Code the script in a file named **rentals.js**

Test your script with 38 people. Now do some hand calculations. How much money did your script say you had to charge per person? \_\_\_\_\_  
If you multiply that out, how much did you collect? \_\_\_\_\_ How much were the vans? \_\_\_\_\_ Why do you have leftover money?

---

---

Make sure to commit your changes and close your project folder when you are finished!

## Pre/Post Increment/Decrement

---

- **JavaScript has four unusual operators**
  - pre-increment
  - post-increment
  - pre-decrement
  - post-decrement
- **Let's start with the pre- and post- increment operator ( ++ )**
  - At their simplest, they add one to a number

### Example

```
var num1 = 1;
var num2 = 1;
var num3 = 1;

// one way to increment a number
num1 = num1 + 1;

// another way to increment a number
++num2;           // pre-increment

// a third way to increment a number
num3++;          // post-increment
```

- **But there is a difference between the pre- and post-increment operators**

- You will see it when you try to use them in a more complex expression

## Pre/Post Increment/Decrement *cont'd*

### Example

```
var x = 5;
var y;

// when you use pre-increment, it does the increment first
// before the other operators
y = ++x;

// increment x first (now x is 6)
// then assign x to y (so y is 6 too)
```

### Example

```
var x = 5;
var y;

// when you use post-increment, it does the increment after
// the other operators
y = x++;

// first assign x to y (so y is 5)
// then increment x (so x is 6)
```

- The same concept would hold with the pre- and post-decrement operator (--)

### Example

```
var num = 5;

--num;           // num is now 4
num--;          // num is now 3
                // no apparent difference between them
```

## Pre/Post Increment/Decrement    *cont'd*

---

### Example

```
var x = 5;
var y;

// when you use pre-decrement, it does the decrement first
// before the other operators
y = --x;

// decrement x first (now x is 4)
// then assign x to y (so y is 4 too)
```

### Example

```
var x = 5;
var y;

// when you use post-decrement, it does the decrement after
// the other operators
y = x--;

// first assign x to y (so y is 5)
// then decrement x (so x is 4)
```

- These operators are mostly used when writing loops

### Example

```
for (var i = 1; i <= 10; i++) {  
}
```

- We will see loops in the next few weeks

# Assignment Operators

---

- The basic JavaScript assignment operator simply moves a value from one variable into another

## Example

```
var x = 5;  
var y;  
  
y = x;
```

- But JavaScript also has assignment operators that also include a math aspect

- For example, `+=` adds the value on the right to the variable on the left

## Example

```
var onHand = 10;  
var qtyReceived = 6;  
  
onHand += qtyReceived;
```

- These type of assignment operators exist for all of the arithmetic operators we've examined so far (ex: `+=`   `-=`   `*=`   `/=`   `%=`)

## Example

```
var onHand = 62;  
var qtyPurchased = 6;  
  
onHand -= qtyPurchased;
```

## Precedence (again)

---

- With what we know, our precedence char has now expanded to include:

| OPERATOR                    | ASSOCIATIVITY |
|-----------------------------|---------------|
| ( )                         | left-to-right |
| ++ --                       |               |
| - (negation) + (unary plus) | right-to-left |
| * / %                       | left-to-right |
| + -                         | left-to-right |
| = += -= *= /= %=            | right-to-left |

Section 2–2

Parsing Strings into Numbers

## Parsing Strings into Numbers

---

- You can't perform mathematical operations on strings

- + performs concatenation when the operands are strings

### Example

```
var x = "123";
var y = "456";
var z = x + y;
console.log(z);      // displays 123456
```

- JavaScript has several functions that can parse a string value and return the numeric equivalent, including

- parseInt() parses a string and returns an integer
  - parseFloat() parses a string and returns a floating point number
  - Number() parses a string and returns a numeric value

- You will need these functions quite a bit when you start building web pages and let users enter data in forms

- Why? Because for fields return strings no matter what type of data the user entered!

## Using parseInt()

- **parseInt()** parses a string and returns an integer

- An integer is a whole number (ex: 7, 1023, or -17)

### Example

```
var hrsWorked = "33";           // a String object  
hrsWorked = parseInt(hrsWorked); // now a Number object
```

- Leading and trailing spaces are allowed

### Example

```
var hrsWorked = " 33  ";  
hrsWorked = parseInt(hrsWorked); // value is 33
```

- If the first character can't be converted to a number, **parseInt()** returns NaN

- You will then have to use an if to detect the issue and respond accordingly

### Example

```
var hrsWorked = "Thirty three";  
hrsWorked = parseInt(hrsWorked);           // value is NaN  
  
if (isNaN(hrsWorked)) {  
    // tell user about the problem with hrsWorked  
}
```

## Using `parseInt()` cont'd

### Example

```
var a = parseInt("10.00");           // a is 10
var b = parseInt("10.33");           // b is 10
var c = parseInt("10 11 12");       // c is 10
var d = parseInt("10 years ago");   // d is 10
var e = parseInt("over 10");         // e is NaN
```

- If the string begins with "0x", it assumes base 16

- Color codes are often specified using base 16, or hexadecimal as it is usually called
  - \* For example, 0xFF0000 is red, 0x00FF00 is green, 0x0000FF is blue, and 0x808080 is gray
- However, we will not have to convert hex into decimal in this class

### Example

```
var a = parseInt("0x3FA2");        // a is 16290 in base 10
```

## Using `parseFloat()`

---

- `parseFloat()` parses a string and returns a floating point number
  - A floating point number can have a decimal point (ex: 7.123, .000505, or -17.2)

### Example

```
var payRate = "10.75";           // a String object
payRate = parseFloat(payRate);    // now a Number object
```

- Leading and trailing spaces are allowed

### Example

```
var payRate = " 10.75 ";          // a String object
payRate = parseFloat(payRate);    // now a Number object
```

- If the first character can't be converted to a number, `parseFloat()` returns NaN

### Example

```
var a = parseFloat("10.00");      // a is 10
var b = parseFloat(".33");        // b is .33
var c = parseFloat("10.5 11.6");  // c is 10.5
var d = parseFloat("10.5 years ago"); // d is 10.5
var e = parseFloat("over 10");    // e is NaN
```

## Using Number ()

- **Number ()** is a little different than the others because it converts different object values to their numbers
  - Strings are parsed to numbers
  - Booleans are parsed to numbers
    - \* true is 1 and false is 0
  - Dates are parsed to numbers (a millisecond value)

### Example

```
var payRate = "10.75";           // a String object
payRate = Number(payRate);       // now a Number object
```

### Example

```
var a = Number("123");          // a is 123
var b = Number("45.67");        // b is 45.67
var c = Number("40 years");     // c is NaN

var d = Number(true);           // d is 1
var e = Number(false);          // e is 0

var f = new Date(2019, 5, 20);   // f is a Date object
var g = Number(f);              // g is 1561006800000
```

## Unary + Also Converts to Number

---

- You can also use the unary + to convert a value to a number
  - It can convert string representations of numbers, as well the values true, false, and null
    - \* true is converted to 1 and false is converted to 0
    - \* null is converted to 0

### Example

```
var payRate = "10.75";           // a String object
payRate = +payRate;              // now a Number object
```

# Exercises

---

Create a subfolder in Mod02 named NumericConversions. These exercises should be placed there.

## EXERCISE 1

Create a script named `conversion_tests.js` file. Add a comment the top the script using the format below:

```
// Description: This script tests various numeric  
//               conversion techniques  
// Author: Jordan Q. Newprogrammer
```

Define the following variables in your script:

```
var a = " 101.1  ";  
var b = "55";  
var c = "402 Stevens";  
var d = "Number 5  ";
```

Now, perform the following tasks:

1. Convert each using `parseInt()` and display the results
2. Convert each using `parseFloat()` and display the results
3. Convert each using `Number()` and display the results
4. Convert each using the unary + operator and display the results

Was the output of each what you expected? Complete the following table with your answers.

| Expression    | <code>parseInt()</code><br>result | <code>parseFloat()</code><br>result | <code>Number()</code><br>result | Unary +<br>result |
|---------------|-----------------------------------|-------------------------------------|---------------------------------|-------------------|
| " 101.1  "    |                                   |                                     |                                 |                   |
| "55"          |                                   |                                     |                                 |                   |
| "402 Stevens" |                                   |                                     |                                 |                   |
| "Number 5  "  |                                   |                                     |                                 |                   |



# Module 3

## Programming with Conditionals

## Section 3–1

Making Decisions with an if/else

## Making Decisions using the `if` Statement

---

- Up until now, all of our logic has been sequential
  - Step 1, step 2, step 3!
- In JavaScript, you can use an `if` statement to perform actions based only when specified conditions are true

### Syntax

```
if ( /* some condition */ )  
    single-statement;
```

- In order to build the condition, you have to use comparison operators
  - Comparison operators are used to see how two values relate to each other

# Comparison Operators

---

- The more commonly used comparison operators are:
  - `==` returns true if the values are equal
  - `!=` returns true if the values not are equal
  - `>` returns true if the first value is greater than the second
  - `>=` returns true if the first value is greater than or equal to the second
  - `<` returns true if the first value is less than the second
  - `<=` returns true if the first value is less than or equal to the second

## Example

```
var numKids = 12;
var price;

price = numKids * 9.95;    // price is $9.95/kid

if ( numKids >= 6 )
    price = price * .9;   // a 10% discount

console.log(price);
```

## Example

```
var amount = 12.50;
var tax = amount * .08;        // 8% sales tax
var taxExempt = false;         // a Boolean variable
...

if ( taxExempt == true )
    tax = 0;
console.log(tax);
```

## Using the `if` Statement

---

- If you want to execute more than one line of code in the `if`, you must place them in between curly brackets

### Syntax

```
if ( /* some condition */ ) {  
    statement(s);  
}
```

```
var numKids = 12;  
var price;  
  
price = numKids * 9.95;      // price is $9.95/kid  
  
if ( numKids >= 6 ) {  
    price = price * .9;      // a 10% discount  
}  
console.log(price);
```

### Example

- Best practice is to always use curly brackets

## Using the `if` / `else` Statement

---

- In JavaScript, your `if` statement can have an `else`

- The `else` specifies an action you want to perform when the `if` statement is false

### Syntax

```
if ( /* some condition */ ) {  
    statement(s);  
}  
else {  
    statement(s);  
}
```

### Example

```
var amount = 12.50;  
var taxExempt = /* value not shown */;  
  
var tax;  
var totalDue;  
...  
  
if (taxExempt == false) {  
    tax = amount * 0.08;          // 8% tax rate  
}  
else {  
    tax = 0;  
}  
  
totalDue = amount + tax;  
console.log(totalDue);
```

## Exercises

---

Create a new folder in your repo named `Workbook2_Mod03`. Then create a subfolder named `ifScripts`. These exercises should be placed there.

### EXERCISE 1

Sketch out a script on paper to calculate gross pay given the variables `payRate` and `hoursWorked`. If the person works more than 40 hours, pay the overtime hours at 1.5 times the rate of regular hours.

When you are finished, review your script with a colleague. Are your algorithms similar? Do you believe each other's code will work?

Open the `MoreMathScripts` folder using Visual Studio Code and create a script named `pay_rules.js`. Code your script. Run it several times with different values for `payRate` and `hoursWorked` and confirm the output is right.

Good test data might be:

| Pay Rate | Hours Worked | Gross Pay | Reason           |
|----------|--------------|-----------|------------------|
| 12.50    | 20           | 250.00    | Under 40 hours   |
| 25.00    | 40           | 1000.00   | Exactly 40 hours |
| 17.30    | 45           | 821.75    | Over 40 hours    |

### (Optional - Challenge!!) EXERCISE 2

Write a script on paper to determine whether a given year is a leap year in the Gregorian calendar. You will need to do a little research to determine what makes a year a leap year.

When you are finished, review your script with a colleague. Are your algorithms similar? Do you believe each other's code will work?

Create a new script in the same folder you used in Exercise 1 named `gregorian_calendar.js`. Code your script. Run it several times with different values for the year. Make sure to test the years 1900, 1950, 1999, 2000, 2001 and 2012.

## Using an if / else / if Statement

---

- Sometimes, you must look through a set of mutually exclusive conditions to make a decision

- You may need to chain your if/else statements

### Syntax

```
if (condition1) {  
    // executed if condition1 is true  
}  
else if (condition2) {  
    // executed if condition1 is false  
    // and condition2 is true  
}  
else {  
    // executed if none of the above is true  
}
```

### Example

```
var name = "Robert";  
var nickname;  
  
if (name == "Andrew") {  
    nickname = "Andy";  
}  
else if (name == "Robert") {  
    nickname = "Bob";  
}  
else if (name == "Cynthia") {  
    nickname = "Cindy";  
}  
else {  
    nickname = name;  
}
```

## Using an **if** / **else** / **if** Statement *cont'd*

---

### Example

| CLASSIFICATION | HOURS EARNED |
|----------------|--------------|
| Freshman       | 0 - 29       |
| Sophomore      | 30 - 59      |
| Junior         | 60 - 89      |
| Senior         | 90+          |

```
var hoursEarned = 103;
var classification;

if (hoursEarned <= 29) {
    classification = "Freshman";
}
else if (hoursEarned <= 59) {
    classification = "Sophomore";
}
else if (hoursEarned <= 89) {
    classification = "Junior";
}
else {
    classification = "Senior";
}
```

# Exercises

---

## EXERCISE 1

Create a script named `greeting.js` in the same `ifScripts` folder you have been using so far. Define a variable that contains the current hour (0- 23). Display one of the greetings below based on the current hour.

| Time              | Greeting      |
|-------------------|---------------|
| until 10:00AM     | Good morning! |
| 10:00AM to 4:59PM | Good day!     |
| 5:00PM or later   | Good evening! |

Remember to run the script several times with different values for the hour.

## EXERCISE 2

Create a script named `complex_taxes.js` in that calculates federal tax based on the values of annual gross income (a number) and a filing status ("Single" or "Joint").

Start by grabbing your gross pay calculation code from the earlier lab and including it here. That code calculates a WEEKLY gross pay. Use that number to estimate ANNUAL gross pay and save it in a new variable.

Now use a series of if statements to determine and save the TAX RATE.

The tax table for single filers is:

| Annual Income Range | Tax Rate |
|---------------------|----------|
| under 12,000        | 5%       |
| 12,000 - 24,999.99  | 10%      |
| 25,000 - 74,999.99  | 15%      |
| 75,000 and over     | 20%      |

The tax table for joint filers is:

| Annual Income Range | Tax Rate |
|---------------------|----------|
| under 12,000        | 0%       |
| 12,000 - 24,999.99  | 6%       |
| 25,000 - 74,999.99  | 11%      |
| 75,000 and over     | 20%      |

## Exercises *cont'd*

---

Use the tax rate to determine the tax withheld from the WEEKLY gross pay. The output of your script might resemble:

```
You worked 45 hours this period.  
Because you earn $10.00 per hour, your gross pay is $475.00  
Your filing status is Single  
Your tax withholdings this period is $47.50  
Your net pay is $427.50
```

To truly test your code, you will need to run it more than once with different values for hours worked, pay rate and filing status.

Examine the tax tables and see if you can determine a set of good test values. Discuss it with your small group. Then test!

## Making And / Or Decisions

---

- Sometimes your decisions are complicated and must incorporate several pieces of information before you can make a decision
  - For example, if I'm in the city of Marietta and the state is Georgia, do one thing. But if the state is Texas, do something else.
- JavaScript has AND ( `&&` ) and OR ( `||` ) operators to help with this
  - When using AND, an expression is true only if both parts are true

### Example

```
var city = /* some value */;  
var state = /* some value */;  
  
if (city == "Marietta" && state == "Georgia") {  
    // do something  
}
```

- When using OR, an expression is true if either part is true

### Example

```
var state = /* some value */;  
  
if (state == "Texas" || state == "Maryland") {  
    // do something  
}
```

# Exercises

---

## EXERCISE 1

Continue working in the `ifScripts` folder. Create a script named `min_max.js` that displays the smallest of three numbers. Then it displays the largest of three numbers.

Name your variables `a`, `b` and `c` and give them values. Then use `if/else` statements to determine and display the answer. Do not use `Math.min()` or `Math.max()`

Change the values in the variables and run the script again.

## EXERCISE 2

Create a script named `show_major.js` that defines two variables for a student: `studentName` and `studentMajor`. The `studentMajor` variable will contain the major code (ex: `CSCI`).

Your script will use look up the student's major code in the table below and displays the name of the major and the location of the department's office. Use the following:

| Major Code | Name of Major    | Department Office       |
|------------|------------------|-------------------------|
| BIOL       | Biology          | Science Bldg, Room 310  |
| CSCI       | Computer Science | Sheppard Hall, Room 314 |
| ENG        | English          | Kerr Hall, Room 201     |
| HIST       | History          | Kerr Hall, Room 114     |
| MKT        | Marketing        | Westly Hall, Room 310   |

Output should resemble the following, although the values displayed will depend upon your variables:

**Student:** Betty  
**Major:** English  
**Advising Location:** Kerr Hall, Room 201

What should your program do if the major code is not one of the ones in the table? Let's let it display `<unknown>` for both major and nothing for advising location.

Run the script several times with different values for name and major code.

## **var vs let**

---

- When JavaScript was first introduced, we used var to declare variables

### **Example**

```
var name = "Dana";
var count = 0;
var message;
```

- ES6 introduced the keywords let and const as a replacement for var
  - a variable defined using let is changeable
  - a variable defined using const is not changeable (const stands for constant)

### **Example**

```
const name = "Dana";
let count = 0;
let message;

count++;                                // valid
message = "Hello " + name;               // valid

name = "Mark";                          // error: name is const
```

- There are some differences between var and let/const with regard to scope

- It isn't significant with the skills you have thus far
- We will talk about scope when we introduce functions

## Section 3–2

Making Decisions with a switch

# The switch Statement

---

- The **switch** statement can be used to make decisions too, but it works differently
  - The value of a variable or expression is compared against values listed in `case` statements
  - If a match is found, it executes the code in the `case` statement

## Syntax

The `break` statement is used to direct flow to the end of the `switch` once the `case` has executed.

```
switch(expression) {  
    case value1:  
        // code block  
        break;  
    case value2:  
        // code block  
        break;  
    default:  
        // code block for none of the above  
}
```

- The biggest limiting factor with using the `switch` is the expression has to evaluate to an integer or character

## Using the `switch`

---

### Example

```
let dayNum = 3;
let dayName;

switch (dayNum) {
  case 0:
    dayName = "Sunday";
    break;
  case 1:
    dayName = "Monday";
    break;
  case 2:
    dayName = "Tuesday";
    break;
  case 3:
    dayName = "Wednesday";
    break;
  case 4:
    dayName = "Thursday";
    break;
  case 5:
    dayName = "Friday";
    break;
  case 6:
    dayName = "Saturday";
    break;
  default:
    dayName = "<unknown>";
    break; // this break isn't "needed"
          // but is included for style
}
```

- The last `case` doesn't have to have a `break` statement

## Using the `switch` cont'd

---

- If more than one case perform the same actions, you can group them together as shown below
  - If the `case` statement doesn't have a `break`, the code will "fall into" the next case

### Example

```
let dayNum = 3;  
let dayName;  
  
switch (dayNum) {  
  case 0:  
  case 6:  
    dayName = "Weekend";  
    break;  
  case 1:  
  case 2:  
  case 3:  
  case 4:  
  case 5:  
    dayName = "Weekday";  
    break;  
  default:  
    dayName = "<unknown>";  
}  
DRAFT
```

## Exercises

---

Create a subfolder named `switchScripts`. These exercises should be placed there.

### **EXERCISE 1**

Write a script named `dept_converter.js` that uses a `switch` statement to determine and print a department name based on a department code.

Assume the following department codes:

- 1      Marketing
- 5      Human Resources
- 10     Accounting
- 12     Legal
- 18     IT
- 20     Customer Relations

Test your script with several different codes.

### **(Optional) EXERCISE 2**

Recode the login from `show_major.js` in a new file called `show_major2.js`. This time, use a `switch` instead of `if` statements.



## **Module 4**

# **JavaScript in the Browser**

## Section 4–1

Building HTML Pages that Use JavaScript

## The <script> Element

---

- Up until now, we have been executing our JavaScript using a Node.js plugin inside of Visual Studio Code
  - This is great for learning the language, but our real goal is to add some behavior to our web pages
- To include JavaScript in a web page, you must use a <script> element to include it
- The <script> element can:
  - contain the script
  - point to an external script file
- Code in script runs when the HTML loading process encounters the script

### Example

We placed the <script> at the bottom of the body in this example so that the messageDiv was created before the script tried to reference it.

```
<html>
  <head>
    <title>Demo</title>
  </head>
  <body>
    <div id="messageDiv"></div>

    <script>
      const messageDiv = document.getElementById("messageDiv");
      messageDiv.innerHTML = "Hello World!";
    </script>
  </body>
</html>
```

# JavaScript Functions

---

- Before we can start adding code to web pages, we need to take a peek at JavaScript functions
- 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`
  - 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 execute  
}
```

## Example

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

## JavaScript Functions *cont'd*

---

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

### Example

```
// This is a call to the function  
showGreetingInConsole();
```

- We use JavaScript functions quite a bit, especially when adding dynamic behavior to a web page

## External Scripts

---

- Instead of embedding the script in the page, you can move it into a function in an external file
  - Code in the HTML page can then "call" the function
- NOTE: If you specify a value for the `src` attribute, the `<script>` element must be empty!

### Example

#### HTML

```
<html>
  <head>
    <title>Demo</title>
  </head>
  <body>
    <div id="messageDiv">
    </div>
    <script src="greetings.js"></script>
  </body>
</html>
```

#### greetings.js

```
"use strict";

function showGreeting() {
  let messageDiv = document.getElementById("messageDiv");
  messageDiv.innerHTML = "Hello World!";
}

showGreeting();
```

- When the script is loaded, any code NOT in a function runs right then -- and in this case the `showGreeting()` function call executes

## Organizing Scripts

---

- Many people like to organize their script files into a subfolder in the web site

### Example

If the site had a subfolder named `scripts`, it would be used in the `<script>` tag's `src` attribute

```
<script src="scripts/greetings.js"></script>
```

- Using a `scripts` folder is considered a best practice

Section 4–2

Interacting with Page Elements

## Accessing Elements on the Page using `getElementById`

---

- To interact with an HTML element programmatically, you must have a reference to it
- `document.getElementById` is the most efficient way to get a reference to an HTML element if the element has an `id` attribute
- Often, we hold the reference in a variable defined using `const`
  - `const` variables that reference form elements can't be changed to reference other elements

### Example

#### HTML

```
<div id="messageDiv"> ... </div>
<input type="text" id="nameField" />
<input type="text" id="ageField" />
<input type="button" id="clickMeBtn" value="Click Me" />
```

#### JavaScript

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

- If an element with the specified `id` doesn't exist on the page, `getElementById` returns `null`

## Working with Contents of an Element using innerHTML

- You can set or get the value of an HTML element using its **innerHTML** property
  - This is used most often with `<p>`, `<span>` or `<div>` elements

### Example

#### HTML

```
<p id="orderStatusPara"></p>
```

#### JavaScript

```
let orderNum = 10254;
let orderStatus = "processing";

let message =
    "Order " + orderNum + "'s status is: " + orderStatus;

const orderStatusPara =
    document.getElementById("orderStatusPara");
orderStatusPara.innerHTML = message;
```

- JavaScript string interpolation can simplify the message

- Use backticks around the string and place variables/values you want inserted between ``${ expression-here }``

#### JavaScript

```
let orderNum = 10254;
let orderStatus = "processing";

let message =
    `Order ${orderNum}'s status is: ${orderStatus}`;

const orderStatusPara =
    document.getElementById("orderStatusPara");
orderStatusPara.innerHTML = message;
```

## Working with <input> Elements

---

- If you want to get or set the value of an <input> element, you must:
  - get a reference to the form field using getElementById
  - use the reference to access the value property
    - \* the value property is a string
- We usually want to process user input after they have had a change to enter the data
  - Often, we process user input when they click a button

### Example

#### HTML

```
<p>Name <input type="text" id="nameField" /></p>
<p>Age <input type="text" id="ageField" /></p>
<p><input type="button" id="showBtn" value="Show" /></p>

<p id="messagePara"></p>
```

#### JavaScript

```
// We would want this code to run when the user clicks the Show button

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

let name = nameField.value;
let age = ageField.value;           // age is a string here

let message =
`Hi ${name}! I hear you are ${age} years old!`;

const messagePara = document.getElementById("messagePara");
messagePara.innerHTML = message
```

## Working with <input> Elements cont'd

- You can also put a value into the element using **value**

- Because the **value** property returns a string, you may have to convert it to a number to use it in a mathematical equation

### Example

#### HTML

```
<p>Year you were born:  
  <input type="text" id="yearBornField" />  
<p>  
  <input type="button" id="findAgeBtn" value="Find Your Age" />  
<p>Age on Dec 31:  
  <input type="text" id="ageField" readonly />  

```

#### JavaScript

```
// We would want this code to run when the user clicks the  
// "Find Your Age" button  
  
let currentYear = 2021;  
  

```

## Working with <input> Elements cont'd

- Any script you created and ran with CodeRunner could be turned into a cool, yet simple, web page

### Example

#### HTML

```
<p>Hours worked: <input type="text" id="hrsWorkedField" /></p>
<p>Pay rate: <input type="text" id="payRateField" /></p>

<p><input type="button" id="calcPayBtn"
       value="Calculate Your Pay" /></p>

<p id="resultPara"></p>
```

#### JavaScript

```
// We would want this code to run when the user clicks the
// "Calculate Your Pay" button

const hrsWorkedField =
  document.getElementById("hrsWorkedField");
const payRateField = document.getElementById("payRateField");

let hrsWorked = Number(hrsWorkedField.value);
let payRate = Number(payRateField.value);

let pay = hrsWorked * payRate;

const resultPara = document.getElementById("resultPara");
resultPara.innerHTML = `Pay is ${pay.toFixed(2)}`;
```

## Combining Finding the HTML Element and Getting the Value

---

- So far, we've shown a two-step process: 1) find the HTML element and then 2) get its value

### Example

```
// Find the fields
const nameField = document.getElementById("nameField");
const ageField = document.getElementById("ageField");

// Get the values
let name = nameField.value;
let age = ageField.value;

// Use the values
let message = name + " is " + age;
```

- But you don't necessarily need to put the value from the HTML element into a separate variable

### Example

```
// Find the fields
const nameField = document.getElementById("nameField");
const ageField = document.getElementById("ageField");

// Use the values
let message = nameField.value + " is " + ageField.value;
```

- You can even get the value and use it in one step if you don't think you'll need a reference to the HTML element

### Example

```
// Use the values
let message = document.getElementById("nameField").value +
  " is " + document.getElementById("ageField").value;
```

Section 4–3

Event Handling

## Events

---

- Events are notifications of things that happen to elements on the web page
- For example:
  - the user has "clicked" a button
  - an input field has lost focus
  - the web page has finished loading
- You can react to these events in JavaScript by writing *event handlers*
  - An event handler is a JavaScript function that executes when the event occurs
- There are several ways to define event handlers and we will see some of them over the next few pages
  - We will see even more as the course progresses

## Event Attributes

---

- **HTML elements have attributes that allow you to assign event handlers**
- **Some of the event attributes include:**
  - `onload` - runs code when the browser has finished loading the page
  - `onclick` - runs code when the user clicks an HTML element
  - `onchange` - runs code when an HTML element has been changed
  - `onfocus` - runs code when the HTML element gains focus
  - `onblur` - runs code when the HTML element loses focus
  - `onkeydown` - runs code when the user presses a keyboard key
  - `onmouseover` - runs code when the user moves the mouse over an HTML element
  - `onmouseout` - runs code when the user moves the mouse off an HTML element

## Coding Event Logic in an HTML Attribute

---

- The HTML event attributes can be assigned code to execute when the event occurs
  - For example, the `onclick` attribute of a form element can specify the code to run when the element is clicked

### Example

```
<input type="button" value="Say Hello"  
      onclick="alert('Hi there!');" />
```

- *However, this is rarely done*

- A *slight* improvement to the code below would be to have the `onclick` attribute call a function

### Example

```
<html>  
<head>  
  <title>Demo</title>  
  <script>  
    "use strict";  
  
    function sayHi() {  
      alert("Hi there!");  
    }  
  </script>  
</head>  
<body>  
  <input type="button" value="Say Hello" onclick="sayHi();" />  
</body>  
</html>
```

- *Although easier to read, code like this is still rarely written*

## Assigning Event Handlers When the Window Finishes Loading

- A common way to associate an event handler to an event is to programmatically assign it when the window finishes loading
  - The `window.onload` event handler executes when the web page has completely loaded all content (including images, script files, CSS files, etc.).

### Example

#### HTML

```
<html>
<head>
    <title>Demo</title>
</head>
<body>
    <input id="helloBtn" type="button" value="Say Hello" />

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

#### JavaScript

```
"use strict";

window.onload = init;

function init() {
    const helloBtn = document.getElementById("helloBtn");
    helloBtn.onclick = onHelloBtnClicked;
}

function onHelloBtnClicked() {
    alert("Hi there!");
}
```

## In-Class Demo

---

Create a subfolder named `HelloWorld` in this module's folder. This demo should be placed there.

Create a web page named `index.html`. Type the following code in the `.html` file:

```
<html>
<head>
    <title>Demo</title>
</head>
<body>
    <input id="helloBtn" type="button" value="Say Hello" />

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

Create a subfolder named `scripts` and add `index.js` to the folder. Type the following code in the `.js` file:

```
"use strict";

window.onload = init;

function init() {
    const helloBtn = document.getElementById("helloBtn");
    helloBtn.onclick = onHelloBtnClicked;
}

function onHelloBtnClicked() {
    alert("Hi there!");
}
```

Browse to your page. When you click the button, do you see the message?

Now, go back and add a `<p>` to the page with the id `messagePara`.

In your script, change the code in `onHelloBtnClicked()` to place the message in `messagePara`.

Re-test!

## **Important Note!**

---

- Because we are handling a button's `onclick` event in these examples, HTML5 validation attribute like `required` or `maxlength` will not work
- We will learn how to make them work in the next workbook
  - Hint: We will have to use submit buttons and handle the form's `onsubmit` event!

# Exercises

---

## EXERCISE 1

Now that you've typed in code exactly as we've shown, let's try writing some simple code by yourselves

Create a subfolder named `GreetByName`. This exercise should be placed there.

This page will provide a form that allows the user to enter their name. When they click a button, the greeting will be personalized for them.

The diagram shows a rectangular web page frame. Inside, there is a text input field with the label "Name" to its left and a rectangular box to its right. Below this is a button with the label "Greet the User".

**Step 1:** Create a web page named `index.html`. Design it like shown above. Both the input field and the button will need an `id`.

```
<p>Name <input type="text" id="nameField" /></p>  
<p><input id="greetBtn" type="button"  
value="Greet the User" /></p>
```

**Step 2:** Create a `scripts` subfolder, then add a script named `index.js` to it.

**Step 3:** In `index.js`, write code to connect the `window.onload` event to an event handler named `init`.

**Step 4:** In the `init()` function, find the button and connect it to an event handler named `onGreetUserBtnClicked`

**Step 5:** Code the `onGreetUserBtnClicked()`. Within it:

- find the name text field using `getElementById`
- extracts the name from the text field
- use the name to create a string that contains "Hello *userNameHere*"
- displays the message using `alert`

## Exercises *cont'd*

---

**Step 6:** Include `index.js` in a `<script>` element at the bottom of the `<body>` element in `index.html`

Finally, test your page. What happens when you enter your name in the input field and click the button?

### EXERCISE 2

Create a subfolder named `Calculator`. This exercise should be placed there.

Create a web page that provides a simple calculator to the user. Design the page as shown below. Give your input fields the ids `number1Field`, `number2Field`, and `answerField`.

Make `answerField` read only in HTML.

The diagram shows a rectangular form with the following layout:  
- Top row: 'Number 1:' followed by an input field, and 'Number 2:' followed by an input field.  
- Middle row: Four buttons labeled 'Add', 'Subtract', 'Multiply', and 'Divide'.  
- Bottom row: 'Answer:' followed by an input field.

Connect the `window.onload` event to an `init` function.

In the `init` event handler, find each button and assign its `onclick` to an event handler. Name the event handlers something like `onAddBtnClicked`, `onSubtractBtnClicked`, etc.

Within the click event handler for each button:

- extract the value from the number 1 input field and convert it to a number using `Number()`
- repeats the process for number 2
- perform the specified math operation
- display the results in the answer field

## **Exercises** *cont'd*

---

NOTE: Try to get the Add button working first and then use it as a pattern for the other three buttons.

Make sure to place all of your JavaScript in a .js file and include it in the page.

DRAFT

## Adding a Little Error Handling

---

- You might recall we said that JavaScript has a value called **NaN**
  - It stands for **not-a-number** and happens when a conversion or math operation results in a value that isn't numeric
- You can test for **NaN** using the **isNaN** function
- Adding logic to your program that detects invalid user input using this technique moves our academic programs closer to real-life programs
- Let's consider the calculator you just finished; it probably contains some HTML similar to this below

### Example

```
<input type="number" id="number1Field">
...
<input type="number" id="number2Field">
...
<input type="button" id="addBtn" value="Add">
...
<input type="text" id="answerField" readonly>
```

- If we want to display a message describing user input errors, we will need to add a **<p>** element

## Adding a Little Error Handling *cont'd*

### Example

```
<input type="number" id="number1Field">
...
<input type="number" id="number2Field">
...
<input type="button" id="addBtn" value="Add">
...
<input type="text" id="answerField" readonly>
...
<p id="messagePara"></p>
```

- After we convert user input to a number, check its value
  - If it is NaN, display a message
  - If it isn't NaN, calculate the answer, display the answer, and then clear any message that might be in messagePara

### Example

```
function onAddBtnClicked() {
    // find the HTML elements
    const number1Field = document.getElementById("number1Field");
    const number2Field = document.getElementById("number2Field");
    const messagePara = document.getElementById("messagePara");

    // get user inputs
    let number1 = Number(number1Field.value);
    let number2 = Number(number2Field.value);

    // check to see if user inputs were invalid
    if (isNaN(number1) || isNaN(number2)) {
        messagePara.innerHTML =
            "One or more of your input values are invalid";
        return; // exit the addBtnClicked function
    }
}
```

## Adding a Little Error Handling *cont'd*

---

```
// display the results
let answer = number1 + number2;
const answerField = document.getElementById("answerField");
answerField.value = answer;

// clear any previous error message
messagePara.innerHTML = "";
}
```

## Mini-Project

---

Create a subfolder named Temperatures. This exercise should be placed there.

In this exercise, you will create three web pages:

- a home page
- a page with a calculator that converts Fahrenheit temperatures to Celsius
- a page with a calculator that converts Celsius temperatures to Fahrenheit

Configure the site structure by:

- adding two paragraphs of lipsum text from <https://www.lipsum.com> to your home page
- adding a nav bar to all three pages that will allow navigation between them

On each calculator page, create an HTML form with a place for an input temperature, a convert button, and a place to put the output temperature.

Use the `window.onload` event on each page to connect the button to an event handler with a good name (ex: `onConvertBtnClicked`). Then, code the button behavior on each form.

Test your pages. Make sure you see an error message if the input temperature isn't a number.

Now add a reset button to each page. Test to make sure it clears your form.

## **Appendix A**

### **Learning References**

# Learning References

---

## **Books**

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

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

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

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

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

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

## **Online References**

### **JavaScript:**

[www.w3schools.com/js](http://www.w3schools.com/js)  
JavaScript tutorial and references

[developer.mozilla.org/en-US/docs/Web/JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript)  
JavaScript tutorial and references

### **jQuery:**

[www.w3schools.com/jquery](http://www.w3schools.com/jquery)  
jQuery tutorial and references

[jquery.com](http://jquery.com)  
jQuery tutorial and references

### **Node.js and Express:**

[www.w3schools.com/nodejs/](http://www.w3schools.com/nodejs/)  
Node.js tutorial and references

[expressjs.com](http://expressjs.com)  
Express guides and references

[developer.mozilla.org/en-US/docs/Learn/Server-side/  
Express\\_Nodejs/Introduction](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction)  
Node.js and Express tutorial and references