JAVASCRIPT

GOALS FOR THIS UNIT

1. Intro to Programming
2. Conditional Logic

OUTPUT

Output is any information provided by the program.
We will use it often when debugging our code.

OUTPUT

`console.log()` is the JS equivalent to a standard out.

This works in the browser and in NodeJS. It will print whatever is contained inside the parentheses to the console.

STANDARD OUT

```
console.log('Hello, World!');
```

produces:

```
Hello, World!
```

ALERTS

We can alert the user to something in a similar manner, using `alert()`.

ALERT

```
alert('Hello, World!');
```

produces:

COMMENTS

Just as in HTML & CSS, we can (and should) use comments in JavaScript.

```
/* Multi-line comments look
like this. */

// Single-line comments look like this.
```

VARIABLES

VARIABLES

Think of variables as boxes that you put stuff inside of. Each box has a label (the variable's name), which gives you some idea of what it holds. JavaScript variables can contain lots of different types of information.

VARIABLES

```
var numberOfStudents = 15;
var firstName = "Drew";
var _lastName = 'Roberts';    // single or double quotes for strings
var $isDetroitCool = true;
var anotherVar2 = 'hooray';
```

DECLARATION

Before we can use a variable, we need to declare it using the `var` keyword.

```
// declare one or more variables individually
var school;
var student;

// declare multiple variables in one statement
var teacher,
    subject,
    period;
```

INITIALIZATION

There's not too much we can do with an empty box. To give our variables some value, we need to initialize them.

```
...

school = "Grand Circus";
teacher = "James";
```

Note: these variables have been declared earlier in our code!

ONE FELL SWOOP

We can declare *and* initialize our variables all at once (as we saw in the first "variables" slide).

```
var andreSays = "Anybody want a peanut?";
```

NAMING IN JAVASCRIPT

1. Must begin with a letter, _, or $
2. Can contain letters, numbers, _, and $
3. Names are case-sensitive

DATA TYPES

DATA TYPES

- Number
- String
- Boolean
- Array
- Object
- Function

NUMBER

The only numeric data type in JavaScript, it can be used to represent integers and floating-point numbers and has three symbolic values `+Infinity`, `-Infinity`, and `NaN` (Not a Number).

```
var myAge = 24;
var priceOfIceCream = 6.25;
```

Note: Be careful not to mix integers with floating-point numbers (which have decimals) when doing arithmetic! You will probably not like what happens.

STRING

A string is a sequence of characters strung together to represent text. Think of a "Happy Birthday!" banner. Each letter is connected by a *string* to create a message humans can read.

```
var myName = "Aisha";
var faveMovie = "The Princess Bride";
```

BOOLEAN

Boolean values are either true or false. They are generally used in our programs' logic.

```
var jsIsFun = true;
var detroitIsScary = false;
```

ARRAY

An array is basically a list of items, each item having what's called an index. The first item in an array has an index of 0 and each subsequent item's index increases by 1, e.g., the 42nd item in an array has an index of 41.

```
var contentTeam = ["Aisha", "James", "Jeseekia", "Kim", "Xinrui"];
var randomStuff = [5, false, 847.3, "puppies", ["another", "array"]];
```

OBJECT

We will talk in much more detail about objects later in the course, but for now think of them as a collection of properties and values.

```
var startup = {
  name: 'Grand Circus',
  city: 'Detroit',
  staffSize: 13
}
```

FUNCTION

We will also spend a lot of time talking about what we can do with functions further down the road. Functions allow us to bundle up parts of our code that we want to use more than once.

```
function sayHello() {
  console.log("WAAASSAAAAAAAPP?!?!?")
}
```

JS DYNAMIC TYPING

```
var yourButts = "hold on";
var yourButts = 121;
var yourButts = true;
var yourButts = ['hold', 'onto', 'your', 'butts'];
var yourButts = {
  youMust: 'hold onto them',
  all: 'of this is legal JavaScript',
  we: 'are reassigning these values to the same variable handle'
};
var yourButts = function() {
  return 'mindblown.gif';
};
```

ARITHMETIC

Arithmetic	operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus	%
Increment	++
Decrement	--

ASSIGNMENT

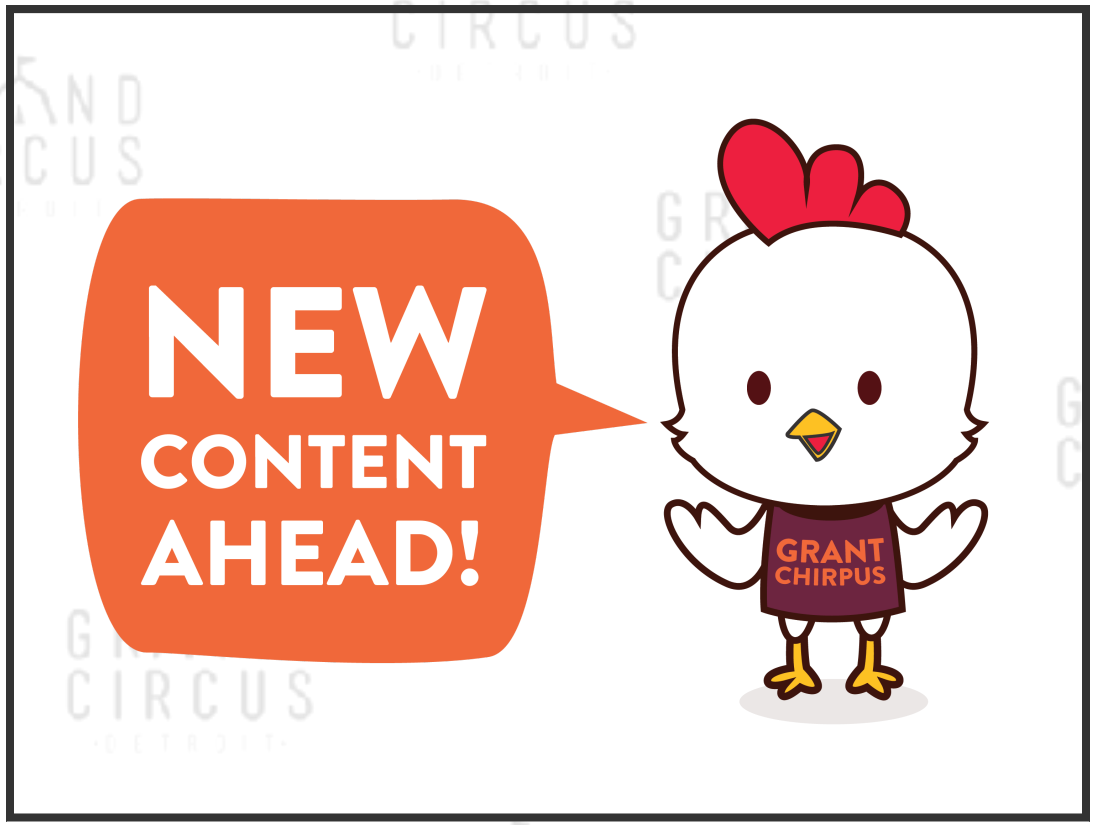
Arithmetic	operator
standard assignment	`=`
plus equals	`+=`
minus equals	`-=`
assignment by multiplication	`*=`
assignment by division	`/=`

STRING CONCATENATION

We can use the plus sign to combine strings.

```
var firstName = 'James ';
var lastName = 'York';
var fullName = firstName + lastName;
console.log(fullName); // > 'James York'

var name = 'j';
name += 'ames';
console.log(name); // > 'james'
```


NEW
CONTENT
AHEAD!
GRANT
CHIRPUS

COMPARISON

COMPARISON

comparison	operator
Equality	`===`
Inequality	`!==`
Greater than	`>`
Greater than or equal to	`>=`
Less than	`<`
Less than or equal to	`<=`

DOUBLE EQUALS

- 'Shallow' equals ==
- 'Shallow' inequals !=

Performs a type coercion before checking equality.

TYPE COERCION

```
true == "true"  // > false
true === "true" // > false
"1" == 1        // > true
"1" === 1       // > false
"1" != 1        // > false
```

Don't use double equals. I'm only showing them because you may see them in the code bases you work on.

SERIOUSLY. DON'T USE THEM.

TRUTHY & FALSY VALUES

WHAT'S IT MEAN?!

Any value in JavaScript can be treated as either true or false.

FALSY

Falsy values are treated as if they are false.

- The actual value `false`
- The number 0
- NaN (Not a Number)
- Empty values
- A variable with no value assigned

TRUTHY

Truthy values are treated as if they are true.

- The actual value `true`
- Nonzero numbers
- Nonempty strings
- Mathmatical expressions

LOGICAL OPERATORS

LOGICAL OPERATORS

&&	AND
||	OR
!	NOT

AND

The logical `AND` returns a value of `true` only if both operands are true. Otherwise, it returns a value of `false`.

OR

The logical `OR` returns a value of `true` if either operand is true. If both are false, it returns `false`.

NOT

The logical `NOT` returns the opposite value of the single operand.

LOGICAL EXPRESSIONS

What will each statement evaluate to?

```
(true && true)
(true && false)
(true || false)
(!true)
```

LOGICAL EXPRESSIONS

```
(true && true)  // > true
(true && false) // > false
(true || false) // > true
(!true)         // > false
```

CONDITIONAL LOGIC

CONDITIONAL LOGIC

if / else statements determine which parts of the code should run under certain conditions. You can probably follow along with the code below. (Don't worry if not!)

```
if (true) {
  // do something
} else if (true) {
  // do something else
} else {
  // do another something else
}
```

IF STATEMENT

The code block between the curly braces of an if statement is only run if the condition evaluates to true. Otherwise, the entire block is skipped and the program continues.

```
if (condition) {
  // do something
}
```

IF/ELSE STATEMENT

Rather than immediately exiting the if statement, we can set some default code which will run if the condition is false using the `else` keyword.

```
if (condtion) {
  // do something
} else {
  // do this other thing
}
```

ELSE IF

We can evaluate more than one condition. If the first condition in the if statement evaluates to false, we can check as many others as we wish using `else if`.

```
if (condition) {
  // do something
} else if (another condition) {
  // do a different thing
} else {
  // do this other thing
}
```

SWITCH STATEMENT

It's easy to complicate things with overuse of `else if`. We can use switch statements if we have several options that each require a unique response.

```
switch (expression) {
  case value1:
    // Do when the result of expression matches value1
    [break;]
  case value2:
    // Do when the result of expression matches value2
    [break;]
  ...
  case valueN:
    // Do when the result of expression matches valueN
    [break;]
  default:
    // Do when none of the values match
    [break;]
}
```

LAB 3

USING THE FUNDAMENTALS

INSTRUCTIONS

Write some JavaScript! It should:

1. Set a temperature as a variable (a Number), assume this is a Fahrenheit temperature.
2. Calculate the temperature conversion to Celsius
3. Log the converted temperature to the console.

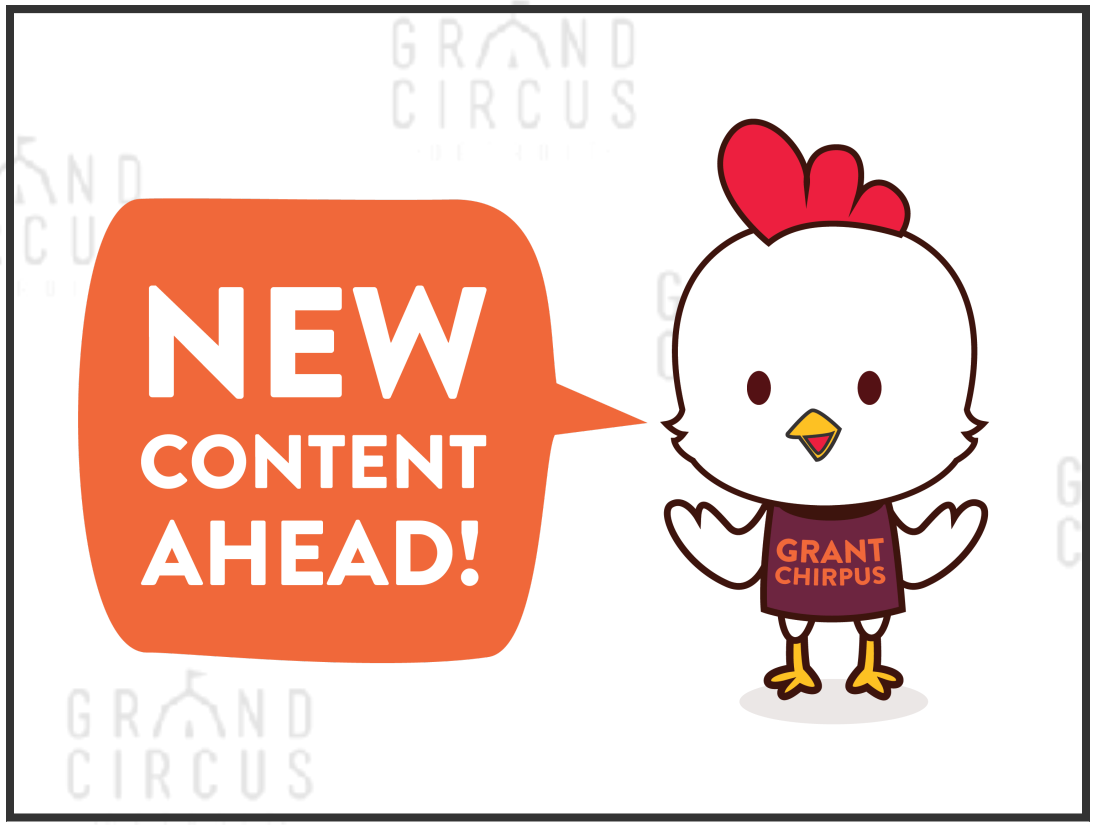
BONUS!

1. Use strings to help the user understand what your program is doing.
2. Add the conversion from Celsius to Fahrenheit.

FIGURE IT OUT

List integers 1-100. Numbers divisible by 3 should be replaced with the word "Fizz", those divisible by 5 replaced by the word "Buzz", and those divisible by both 3 and 5 replaced with the word "FizzBuzz".

DEMOS


NEW
CONTENT
AHEAD!
GRANT
CHIRPUS

LOOPS

GOALS FOR THIS UNIT

1. Review
2. Loops
3. Intro to Functions

REVIEW

LOOPS

LOOPS

Loops allow us to reiterate (execute multiple times) over sections of code based on a condition we set. From this point on, when I say "iteration", I mean executing the code inside of a loop one time.

There are a few different types of loops. The loop you use will depend on the task you're trying to accomplish.

```
for(var i=0; i < n; i++) {
  // repeating things
}

while(true) {
  // repeating things
}

do {
  // repeating things
} while(true);
```

LOOPS

In practice, I usually use for loops or a more specialized iterator like `forEach()` (more on that later).

FOR LOOP

For loops have three optional expressions, enclosed in parentheses and separated by semicolons, followed by a statement or a set of statements executed in the loop.

```
for ([initialization]; [condition]; [final-expression]) {
   // repeating things
}
```

It is common to use a counter to determine the number of times a program should run through a for loop. We use `i` as this counter's variable name by convention.

INITIALIZATION

The first expression assigns a value to a variable, usually a counter.

```
for (i = 0; [condition]; [final-expression]) {
   // repeating things
}
```

CONDITION

The second expression is evaluated before the loop is run. If it evaluates to true, we run the statement(s) inside. If it is false, we break out of the loop and continue executing the program.

```
for ([initialization]; i < n; [final-expression]) {
   // repeating things
}
```

FINAL EXPRESSION

The third and final expression in parentheses is run at the end of each iteration of the loop. It is primarily used to update the counter.

```
for ([initialization]; [condition]; i++) {
   // repeating things
}
```

QUICK EXERCISE

In jsbin.com, build a `for` loop that counts from 1 to 10. Print each number to the console. (10 mins.)

WHILE LOOP

A while loop reiterates as long as a given condition evaluates to true. The condition is evaluated before each iteration of the loop.

```
while (condition) {
  // repeating things
}
```

The block inside the while loop must provide a way to exit the loop. Otherwise, we get caught in an infinite loop that will continue to run over and over again.

```
var a = 0;
var j = 0;

while (a < 30) {
  a++;
  j += a;
}
```

QUICK EXERCISE

In jsbin.com, build a while loop that produces the following output. (10 min)

For fun, remove whatever counter you used to stop the loop to force an infinite loop.

```
You are 10 years old!
You are 11 years old!
You are 12 years old!
You are 13 years old!
You are 14 years old!
You are 15 years old!
You are 16 years old!
You are 17 years old!
You are legally an adult!
```

DO...WHILE LOOP

A do...while loop is very similar to a while loop, but code contained within the loop is always run at least once because the condition is evaluated at the end of each iteration, rather than the beginning.

```
do
 // repeating things
while (condition);
```

WHAT IS IT GOOD FOR?

Suppose we want to ask the user a question over and over until we get an acceptable answer. We'd end up having to repeat code using a while loop. Using do...while instead allows us to cut down on the code we write.

```
var faveFruit = alert("What's your favorite fruit?");

while (faveFruit !== "apples") {
  faveFruit = alert("What's your favorite fruit?");
}
```

```
var faveFruit;

do
  faveFruit = alert("What's your favorite fruit?");
while (faveFruit !== "apples");
```

FUNCTIONS

FUNCTIONS

Functions allow us to reuse parts of our code. They are like the verbs of JavaScript.

```
function beAwesome() {
  // go on being awesome
}

beAwesome();
```

THE BREAKDOWN

1. We begin declaring functions by using the `function` keyword.

```
function
```

THE BREAKDOWN

2. Optionally, we can name our functions.

```
function nameOfFunction
```

THE BREAKDOWN

3. The declaration of the function must include a set of parentheses.

```
function nameOfFunction()
```

THE BREAKDOWN

4. The code we want to be able to use in multiple places goes inside a set of curly braces.

```
function nameOfFunction() {
  // things the function does
}
```

THE BREAKDOWN

5. Executing, or *calling*, the function is done by typing the name of the function followed by parentheses.

```
function nameOfFunction() {
  // things the function does
}

nameOfFunction();
```

THE BREAKDOWN

6. Optionally, we can give the function one or more parameters. We can then pass arguments to the function to change the way it works.

```
function nameOfFunction(someParameter) {
  // things the function does, using someParameter
}

nameOfFunction(someArgument);
```

QUESTIONS?

LAB 4

TEMPERATURE CONVERTER AS A FUNCTION

INSTRUCTIONS

1. Write a function that converts temperature.
2. Your function should accept two arguments: a temperature and a string indicating what unit to convert to (ex. "C" or "F").
3. Call the function with the following arguments to verify your outputs.

```
convertTemp(212, "C"); // > 100
convertTemp(32, "C");  // > 0
convertTemp(65, "C"); // > 18.333
convertTemp(-40, "F"); // > -40
```

FIGURE IT OUT

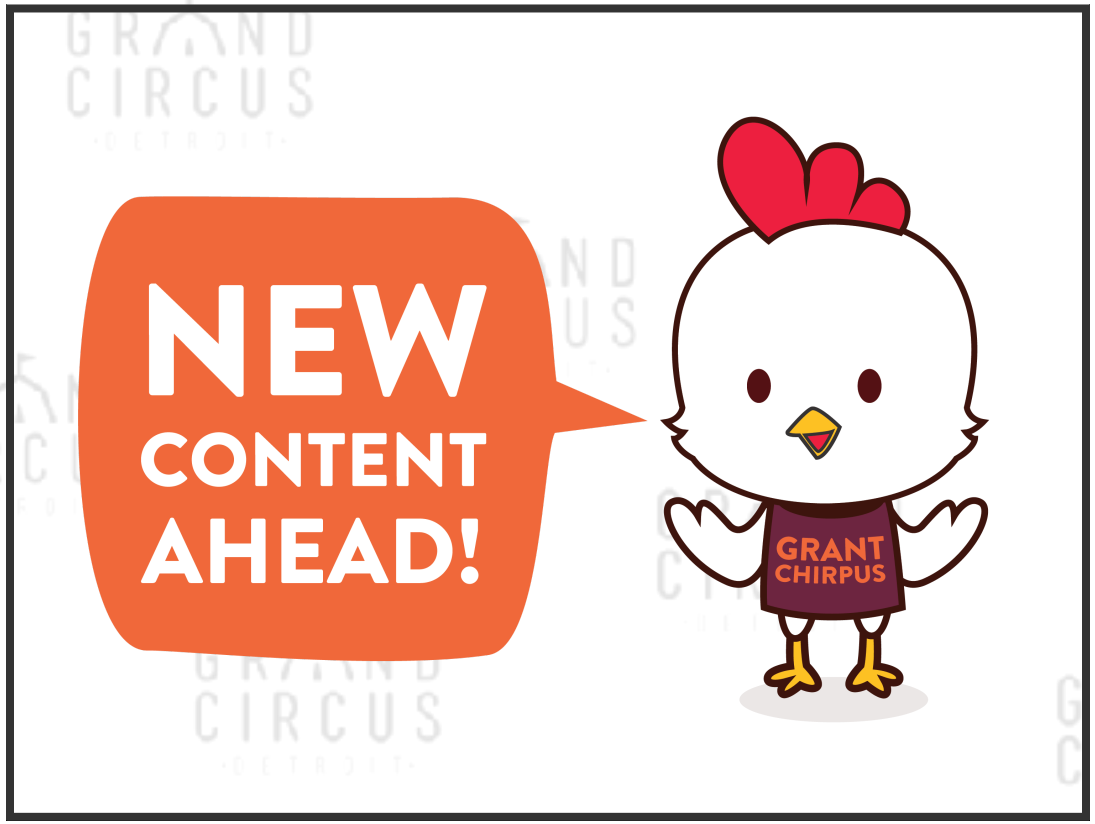
Write a function that assigns a random integer between 1 and 10 to a variable. Prompt the user to input a guess number. If the user input matches with the random number, print a "Good Work" message. Otherwise, display a "Not a Match" message and prompt the user to guess again.

BONUS: If the user's guess is not a match. Give a hint like "Nope, higher" or "Lower" based on the user's guess in relation to the target number.

SUGGESTED READING

From JavaScript & jQuery:

- Chapter 2: 70-73,
- Chapter 3: 100-144


NEW
CONTENT
AHEAD!
GRANT
CHIRPUS

ARRAYS, METHODS, & OBJECTS

GOALS FOR THIS UNIT

1. Review
2. Arrays
3. Built-in methods
4. Objects in JavaScript

REVIEW

DEMOS

DISCLAIMER

I realize that this is a lot of information. It's okay if you feel like this:

ARRAYS

ARRAYS

In JavaScript, an array is a data structure consisting of a collection of elements (values or variables), each identified by at least one array index or key.

ARRAYS

Arrays in JavaScript are special because the individual elements can be anything that can be stored in a variable.

ARRAY DEMO

```
// A perfectly legal array in JavaScript
var myArray = [
  'thing',
  12,
  ['another', 'array'],
  { my: 'object' },
  function() {}
];
```

ARRAYS

Granted, *most* of the time arrays are collections of like elements, but this is a feature that is used all over JS i.e. function argument lists.

ARRAYS

Fortunately, other array functionality works as expected (if you've worked in other languages before).

```
var coolColors = [
  'green',
  'blue',
  'purple'
];

coolColors.length // 3 (number of items in array)
```

ARRAYS

Bracket notation can also be used to reassign elements in an array OR add new elements.

```
var coolColors = [
  'green',
  'blue',
  'purple'
];

coolColors[1] = 'lavender';
coolColors[3] = 'indigo';
coolColors.length; // > 4
```

ZERO-BASED INDEXING

Notice anything unexpected in that last example?

Assigning a new color to index 3 increased the length of our 3-item `coolColors` array. That's because indexing begins with 0.

```
var coolColors = [
  'green',
  'blue',
  'purple'
];

coolColors[0]; // > 'green'
```

The `length` property returns the length of an object. E.g., if the object is an array, the number of elements is returned; if a string, the number of characters.

```
var rainbow = ["red", "orange", "yellow", "green", "blue"];
var name = "Aisha";
rainbow.length // > 5
name.length // > 5
```

METHODS

WHAT'S A METHOD?

Methods are very similar to functions, but they have a more specific purpose. Methods allow objects (which we'll talk about in more depth very soon) do things or have things done to them.

USES

Often, methods will be used to do the following:

- Return some information about the object
- Change something about the object

BUILT-IN METHODS

There are a number of methods given to us for free by the JavaScript language. You don't need to understand exactly *how* they work for right now, but you should know how to make affective use of them.

We'll go over several examples:

The `toString()` method is used to convert many other types of data into a string that represents that data.

```
var ageAtHeart = 5;
var ageAsString = ageAtHeart.toString(); // > "5"
```

The `charAt()` method returns the character at a specified index of a string.

```
var faveThings = "raindrops on roses";
var zeroIn = faveThings.charAt(0); // > "r"
```

The `concat()` method is used to combine multiple strings into one. It can also be used to join arrays together.

```
var faveThings = "raindrops on roses";
var badThings = " when the dog bites";
var newString = concat(faveThings, badThings);
// > "raindrops on roses when the dog bites"
```

The `indexOf()` method returns the first index at which the specified value occurs.

```
var faveThings = "raindrops on roses";
var rainbow = ["red", "orange", "yellow", "green", "blue"];
faveThings.indexOf("r"); // > 0
rainbow.indexOf("orange"); // > 1
```

The `pop()` method removes the last element in an array and returns it.

```
var rainbow = ["red", "orange", "yellow", "green", "blue"];
rainbow.pop(); // > "blue"
rainbow; // > "red", "orange", "yellow", "green"
```

The `push()` method adds one or more elements to the end of an array and returns the updated length of the array.

```
var rainbow = ["red", "orange", "yellow", "green", "blue"];
rainbow.push("indigo", "violet"); // > 7
rainbow; // > "red", "orange", "yellow", "green", "blue", "indigo", "violet"
```

The `shift()` method removes the first element in an array and returns it.

```
var rainbow = ["red", "orange", "yellow", "green", "blue"];
rainbow.shift(); // > "red"
rainbow; // > "orange", "yellow", "green", "blue"
```

The `unshift()` method adds one or more elements to the beginning of an array and returns the updated length of the array.

```
var rainbow = ["red", "orange", "yellow", "green", "blue"];
rainbow.unshift("pink"); // > 6
rainbow; // > "pink", "red", "orange", "yellow", "green", "blue"
```

The `forEach()` method accepts a *function* as an argument. That functions accepts an element from the array as an argument. Then the body of that function is executed *for each* (see what they did there?) element in the array. `forEach()` is an even safer option for iterating over an array than a regular `for` loop.

```
var rainbow = ["red", "orange", "yellow", "green", "blue"];
rainbow.forEach(function(element){
  console.log(element)
});

"red", "orange", "yellow", "green", "blue"
```

Function as an argument say whaaat?!

STOP! White board time!

FINDING METHODS

There is no point in trying to memorize all of this nonsense. You MUST learn to FIND the answers you seek! See this giant list of methods and try some of them out.

OBJECTS

Objects are a data structures that allow us to store collections of data including properties (variables) which can include arrays, other objects, or functions.

OBJECTS

```
var myInfo = {
  name: 'James',
  age: 36,
  married: true,
  likes: ['Mythbusters', 'Jim Butcher', 'JavaScript'],
  family: [
    {
      name: "Rebecca",
      relation: "spouse"
    },
    {
      name: "Evangeline",
      relation: "daughter"
    },
    {
      name: "Josephine",
      relation: "daughter"
    },
  ],
  listFamilyMembers: function() { }
};
```

OBJECTS

Object properties can be accessed using dot notation.

```
var name = myInfo.name;
var age = myInfo.age;
var maritalStatus = myInfo.married;
var familyMembers = myInfo.listFamilyMembers();
```

OBJECTS

You can also use dot notation to add new properties to objects.

```
myInfo.hobbies = ['video games', 'quadcopters', 'volunteering'];
myInfo.pets = 4;
```

LAB 5

SHOPPING LIST WITH OBJECTS AND ARRAYS

INSTRUCTIONS

Write a simple shopping list program. We'll build on this in the next lab.

1. Create several grocery item objects with properties of name and price.
2. Store the grocery item objects in an array.
3. Loop through the array and print out the name and price on a new line.
4. Total up the amount with a label ‘total’.

Be prepared to demo your work.

FIGURE IT OUT

Write a JavaScript function to generate an array. The elements in the array should be integers in a range between two integers given as arguments.

```
makeArray(-4, 2);
>> [-4, -3, -2, -1, 0, 1, 2]
```

SUGGESTED READING

From JavaScript & jQuery:

- Chapter 3: 98-99
- Chapter 5: 183-242
- Chapter 6: 243-292

FUNCTIONS, SCOPE, & THE DOM

GOALS FOR THIS UNIT

1. Review
2. More on Functions
3. Variable Scope
4. The Document Object Model
5. JavaScript Events

REVIEW

DEMOS

MORE ON FUNCTIONS

FUNCTIONS

Functions are the special sauce that makes JavaScript such a cool language. Functions in JavaScript are first class objects, meaning:

- A function is an instance of the Object type
- A function can have properties and has a link back to its constructor method
- You can store the function in a variable
- You can pass the function as a parameter to another function
- You can return the function from a function

Let's look at a few of these.

FUNCTIONS

You can store a function in a variable.

```
function feedDog() {
  return "Kibble, canned food, and water";
}

var eveningChores = feedDog;

eveningChores();

// Or you can do this directly with an anonymous function

var feedDog = function() {
  return "kibble, canned food, and water";
};
```

FUNCTIONS

You can pass a function to a function as a parameter.

```
function doEveningChores(chores) {
  chores.forEach(function(chore){
    chore();
  });
}

doEveningChores([feedDog]);
```

FUNCTIONS

WTF was that forEach() thinger?! Glad you asked.

Not only can you pass functions as arguments, you can define them in-line like any other data type literal.

forEach is a method on the Array object that takes a function as an argument. That function is called on each element of the array receiving *it* as an argument.

...you can do the same with objects and arrays

FUNCTIONS

You can return functions from functions

```
function tonightChores(){
  return feedDog;
}

var tonight = tonightChores();
tonight();
```

VARIABLE SCOPE

LOCAL SCOPE

A variable declared within a function has local scope. It is only available to the function in which it's declared.

GLOBAL SCOPE

Variables with global scope are declared outside of a function. They can be accessed anywhere, but can cause problems with bigger, more complex applications. They take up memory and may cause namespace conflicts. (Bad things happen when two entirely separate variables have the same name.)

GLOBAL VS. LOCAL

Variables can have a global or a local scope.

- Variables in a local scope can access global variables.
- Global variables cannot access local variables.

VARIABLE SCOPE

```
var one = 1;

function doStuff() {
  console.log(one);
  var meaningOfLife = 42;
}

doStuff();
console.log(meaningOfLife);

// > 1
// > Syntax error: meaningOfLife is not defined.
```

VARIABLE SCOPE

Key difference - there is no block level scoping

- What would you normally expect the output of this code to be?

```
var meaningOfLife = 0;
function doStuff() {

  console.log(meaningOfLife);
  if(true) {
    var meaningOfLife = 42;
  }
}

// > undefined (Note: not a syntax error and not 0? Why?)
```

DOCUMENT OBJECT MODEL (DOM)

DOM

The document object model (DOM) is an interface which allows programs and scripts to dynamically access and update the content, style and structure of an HTML document.

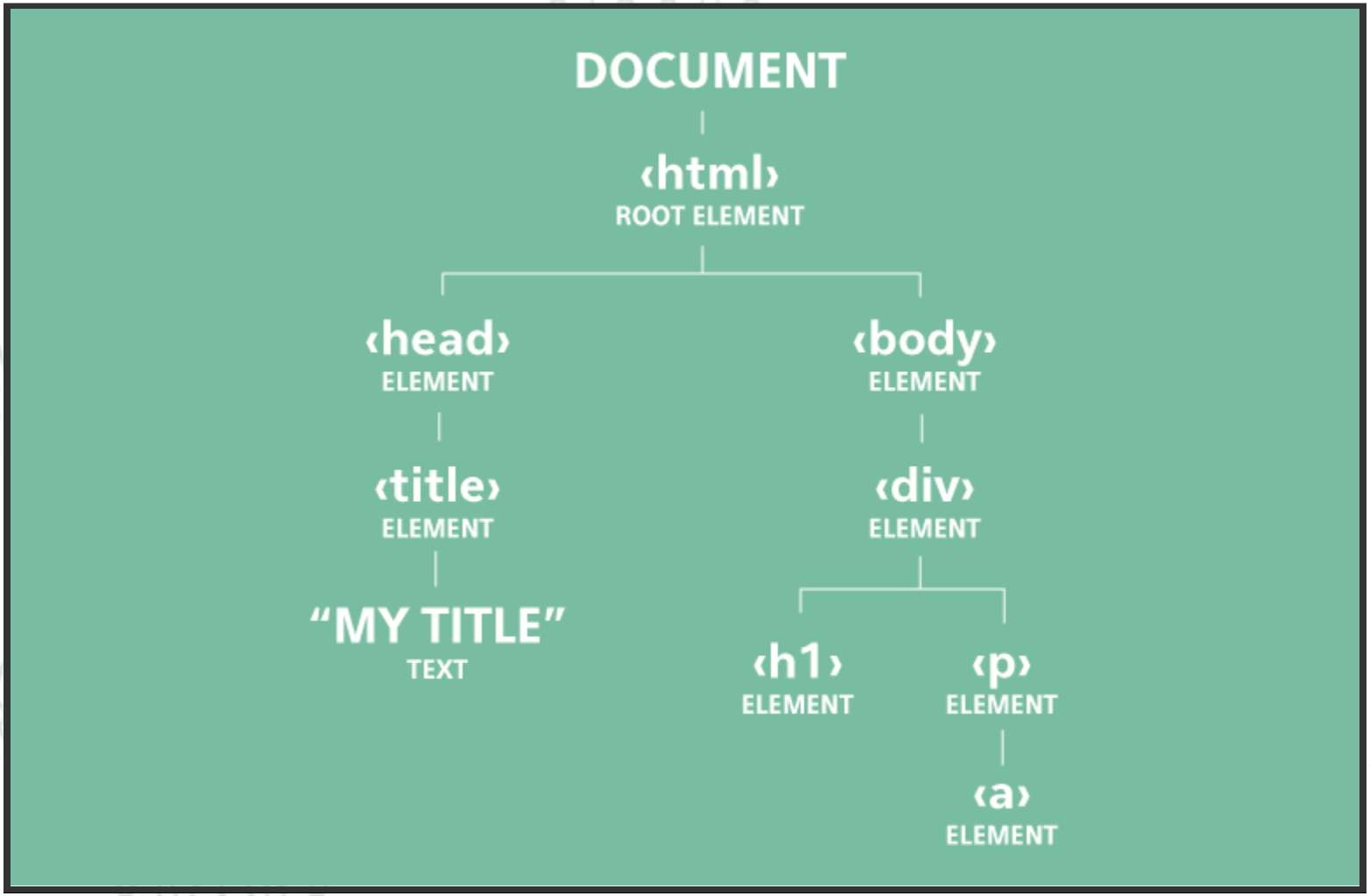
DOM

The DOM is a W3C (World Wide Web Consortium) standard and includes the Core DOM, XML DOM, and HTML DOM. The HTML DOM is a standard model for HTML documents and defines how to get, change, add, or delete HTML elements.

DOM

DOM

The DOM is a tree structure and identifies objects (elements) using nodes



STORING DOM OBJECTS

```
// body element
var bodyNode = document.body;

// html element
var htmlNode = document.body.parentNode;

// Array of all bodies children
var childNodes = document.body.childNodes;
```

DOM

The DOM exposes a number of methods that are used to manipulate the structure of a page. Open any web page in your browser, open your developer tools, and run this command in the console. Each web page loaded in the browser has its own `document` object.

```
document.write('I changed the whole page! #rekt');
```

It’s possible to find elements on the HTML page by parent, sibling, or child node, but that’s time consuming and will make you crazy.

DOM

The better method is to use some of the provided DOM methods by tag, class, or ID.

```
<li class="food">Pizza</li>
<li class="food">Sushi</li>
<li class="food" id="favorite">Schwarma</li>
```

```
var listItems = document.getElementsbyTagName('li');       // Array
var listItems = document.getElementsbyClassName('food');   // Array
var favItem = document.getElementbyId('favorite');         // Object
```

DOM

The attributes of HTML elements can also be accessed and modified through the DOM

```
var img = document.getElementById('myImage');

img.getAttribute('src');
img.setAttribute('src', './images/newImage.jpg');
```

DOM

You can create nodes using the DOM and manipulate their contents using the `innerHTML` property. You can add them to a page by using `appendChild`.

```
var newElement = document.createElement('div');

newElement.innerHTML = '<h1>Hi Everybody!</h1> <p>Hi Dr. Nick!</p>';
document.body.appendChild(newElement);
```

QUESTIONS?

LAB 6

MANIPULATING THE DOM

INSTRUCTIONS

Extend the shopping list program from the last lab.

1. Set up a basic HTML page.
2. Append the items and their prices from the shopping list to the page.
3. Show the total somewhere on the page.

BONUS!

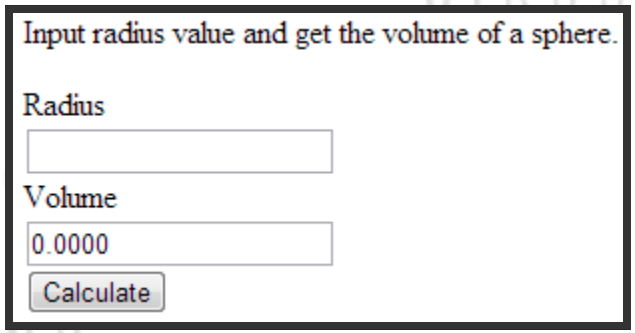
Add a form with text inputs for Name and Price and a button that allows you to add elements to the shopping list.

- Clicking 'Add' updates the list on the page.
- Clicking 'Add' also updates the total.

Be prepared to demo your work.

FIGURE IT OUT

Write a JavaScript program to calculate the volume of a sphere from a user's input. Include appropriate error messages as alerts if the input is a negative number or not a number at all.



SUGGESTED READING

MDN's Intro to Object-Oriented JavaScript

From JavaScript & jQuery:

- Chapter 3: 97
- Chapter 10: 456-457