GRAND CIRCUS

·DETROIT·

## JAVASCRIPT

## WELCOME!

## GOALS FOR THIS UNIT

1. Review Pre-work
2. Intro to Programming
3. Conditional Logic

## INTRODUCTION

DREW ROBERTS

email: jdrewsdc@gmail.com | GitHub: AndrieuxReabeurts

## INTRODUCTION

AISHA BLAKE

email: aisha@grandcircus.co | GitHub: AishaBlake

## REVIEW

## DEMOS

## OUTPUT

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

## OUTPUT

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

```
ello, World!');
```

produces:

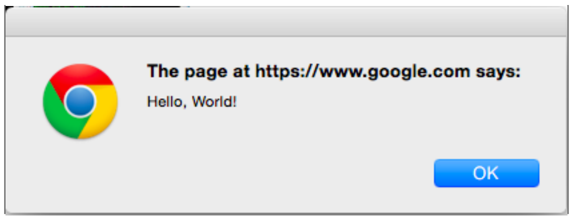
## ALERTS

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

## ALERT

```
World!');
```

produces:



The page at https://www.google.com says:

Hello, World!

## COMMENTS

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

```
comments look

 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

```
udents = 15;
= "Drew";
= 'Roberts';    // single or double quotes for strings
Cool = true;
2 = 'hooray';
```

## DECLARATION

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

```
 or more variables individually


tiple variables in one statement
```

## INITIALIZATION

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

```
d Circus";
es";
```

**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).

```
= "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 , , and (Not a Number).

```
;

Cream = 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.

```
Aisha";
= "The Princess Bride";
```

## BOOLEAN

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

```
true;
cary = false;
```

## ARRAY

An array is basically a list of items, each item having what's called an index. The firs 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.

```
m = ["Aisha", "James", "Jeseekia", "Kim", "Xinrui"];
f = [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.

```
{
 Circus',
it',
3
```

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 us more than once.

```
llo() {
"WAAASSAAAAAAAPP?!?!?")
```

## JS DYNAMIC TYPING

```
= "hold on";
= 121;
= true;
= ['hold', 'onto', 'your', 'butts'];
= {
ld onto them',
s is legal JavaScript',
ssigning these values to the same variable handle'

= function() {
blown.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.

```
= 'James ';
 'York';
 firstName + lastName;
llName); // > 'James York'

;
;
me); // > '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

```
   // > false
 " // > false
   // > true
   // > false
   // > 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
- 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
- Nonzero numbers
- Nonempty strings
- Mathmatical expressions

## LOGICAL OPERATORS

## LOGICAL OPERATORS

AND

---

OR

---

NOT

## AND

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

## OR

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

## NOT

The logical returns the opposite value of the single operand.

## LOGICAL EXPRESSIONS

What will each statement evaluate to?

)
)

## LOGICAL EXPRESSIONS

```
  // > true
) // > false
) // > 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!)

```
ing
e) {
ing else

r something else
```

## IF STATEMENT

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

```
 {
ing
```

## 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 keyword.

```
{
ing

ther 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 .

```
 {
ing
ther condition) {
erent thing

ther thing
```

## SWITCH STATEMENT

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

```
sion) {
 the result of expression matches value1

 the result of expression matches value2


 the result of expression matches valueN

 none of the values match
```

## LAB 1

USING THE FUNDAMENTALS

## SETUP

In a new folder, create a new project (index.html, script.js). Create a base HTML skeleton (no content). Add this line to your tag:

```
cript.js"></script>
```

To make sure it's working, add the line to your script.js and load your page. If you get an alert pop up, you're okay to proceed. If not, your JS isn't being loaded.

## 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.

```
< n; i++) {
things


things


things
```

## LOOPS

In practice, I usually use for loops or a more specialized iterator like (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.

```
zation]; [condition]; [final-expression]) {
g things
```

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

## INITIALIZATION

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

```
ondition]; [final-expression]) {
g 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.

```
zation]; i < n; [final-expression]) {
g things
```

## FINAL EXPRESSION

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

```
zation]; [condition]; i++) {
g things
```

## QUICK EXERCISE

In jsbin.com, build a 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 i
evaluated before each iteration of the loop.

```
on) {
 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.

```
{
```

## 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

```
rs old!
rs old!
rs old!
rs old!
rs old!
rs old!
rs old!
rs old!
y an adult!
```

## DO...WHILE LOOP

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

```
things
on);
```

## 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.

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

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

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

## FUNCTIONS

## FUNCTIONS

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

```
some() {
ng awesome
```

## THE BREAKDOWN

1. We begin declaring functions by using the ______ keyword.

## THE BREAKDOWN

2. Optionally, we can name our functions.

```
fFunction
```

## THE BREAKDOWN

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

```
fFunction()
```

## THE BREAKDOWN

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

```
fFunction() {
e function does
```

## THE BREAKDOWN

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

```
fFunction() {
e function does


();
```

## 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.

```
fFunction(someParameter) {
e function does, using someParameter


(someArgument);
```

## QUESTIONS?

## LAB 2

ITEMIZED SHOPPING LIST

## INSTRUCTIONS

Write a program that prints an itemized shopping list and total to the console.

1. Create variables to hold strings with product names and numbers with product prices.
2. Print out the name and price of each item on a new line.
3. Print the total of all prices with a label ‘total’.

Be prepared to demo your work.

## 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 randon 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.

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 tha
can be stored in a variable.

## ARRAY DEMO

```
 legal array in JavaScript
[


'array'],
t' },
}
```

## ARRAYS

Granted, *most* of the time arrays are collections of like elements, but this is a featur 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).

```
 = [



gth // 3 (number of items in array)
```

## ARRAYS

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

```
 = [




= 'lavender';
= 'indigo';
gth; // > 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 array. That's because indexing begins with 0.

```
= [




// > 'green'
```

The  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.

```
["red", "orange", "yellow", "green", "blue"];
sha";
 // > 5
 > 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 [ ] method is used to convert many other types of data into a string tha
represents that data.

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

The  method returns the character at a specified index of a string.

```
 = "raindrops on roses";
aveThings.charAt(0); // > "r"
```

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

```
 = "raindrops on roses";
= " when the dog bites";
= concat(faveThings, badThings);
s on roses when the dog bites"
```

The [ ] method returns the first index at which the specified value occurs.

```
 = "raindrops on roses";
["red", "orange", "yellow", "green", "blue"];
exOf("r"); // > 0
f("orange"); // > 1
```

The ______ method removes the last element in an array and returns it.

```
["red", "orange", "yellow", "green", "blue"];
 // > "blue"
"red", "orange", "yellow", "green"
```

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

```
["red", "orange", "yellow", "green", "blue"];
indigo", "violet"); // > 7
"red", "orange", "yellow", "green", "blue", "indigo", "violet"
```

The ______ method removes the first element in an array and returns it.

```
["red", "orange", "yellow", "green", "blue"];
); // > "red"
"orange", "yellow", "green", "blue"
```

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

```
["red", "orange", "yellow", "green", "blue"];
t("pink"); // > 6
"pink", "red", "orange", "yellow", "green", "blue"
```

The ______ method accepts a *function* as an argument. That functions accepts a
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. ______ is an even safer option for iterating over an array than a regular ___ loop.

```
["red", "orange", "yellow", "green", "blue"];
h(function(element){
element)
```

```
", "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

```
',

e,
hbusters', 'Jim Butcher', 'JavaScript'],


ebecca",
: "spouse"


vangeline",
: "daughter"


osephine",
: "daughter"


mbers: function() { }
```

## OBJECTS

Object properties can be accessed using dot notation.

```
nfo.name;
fo.age;
tus = myInfo.married;
ers = myInfo.listFamilyMembers();
```

## OBJECTS

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

```
 = ['video games', 'quadcopters', 'volunteering'];
4;
```

## LAB 3

SHOPPING LIST WITH OBJECTS AND ARRAYS

## INSTRUCTIONS

Extend the shopping list program.

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 b
integers in a range between two integers given as arguments.

```
2);

, -1, 0, 1, 2]
```

## 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.

```
og() {
le, canned food, and water";


res = feedDog;

);

do this directly with an anonymous function

function() {
le, canned food, and water";
```

## FUNCTIONS

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

```
ningChores(chores) {
ch(function(chore){




s([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

```
htChores(){
og;


tonightChores();
```

## 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 tak 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

```
ff() {
one);
fLife = 42;


aningOfLife);
```

```
ror: 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?

```
ife = 0;
ff() {

meaningOfLife);

gOfLife = 42;
```

(Note: not a syntax error and not 0? Why?)

## QUESTIONS?

NEW CONTENT AHEAD!
GRANT CHIRPUS

## DOCUMENT OBJECT MODEL (DOM)

## DOM

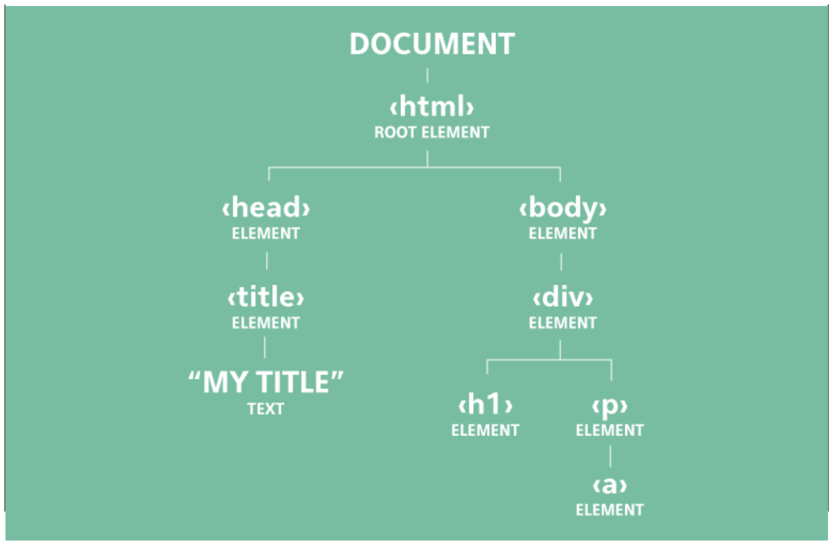
The document object model (DOM) is an interface which allows programs and scripts to dynamically access and update the content, style and structure of an HTM document.

## 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 HTM documents and defines how to get, change, add, or delete HTML elements.

## DOM

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



## STORING DOM OBJECTS

```
t
 document.body;

t
 document.body.parentNode;

l bodies children
 = document.body.childNodes;
```

## DOM

The DOM exposes a number of methods that are used to manipulate the structure o
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
object.

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

It’s possible to find elements on the HTML page by parent, sibling, or child node, bu
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.

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

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

## DOM

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

```
ment.getElementById('myImage');

te('src');
te('src', './images/newImage.jpg');
```

## DOM

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

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

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

## QUESTIONS?

## LAB 4

MANIPULATING THE DOM

## INSTRUCTIONS

Extend the shopping list program from the last lab even further.

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!

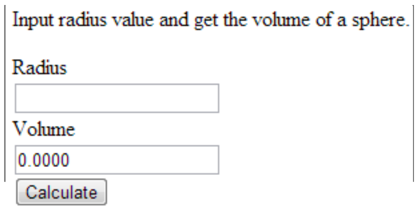
Add a form with text inputs for Name and Price and a button that allows you to add elements to the shopping list.

- 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 no a number at all.


Input radius value and get the volume of a sphere.

Radius

Volume

0.0000