JAVASCRIPT

Making a website up until now has just been writing HTML and CSS! One of the things that makes websites so responsive is that they use not only html/css, but they also use javascript (that's what works so well with bootstrap)! Before we start integrating javascript into our websites, first we'll need to understand how javascript works!

Our first order of business is to understand how javascript works we'll need to use the console! Over in codepen, we can use the javascript window to work with javascript, but instead of rendering code to the site (like we've done before), for now we'll only be using the console!

Primitives:

Javascript is very different from HTML/CSS, it works completely differently, but don't worry! Everything you need to know will be here!

To start out, primitive types are:

Integers: 1,2,3,4,5,ect. **Floating points:** 2.3, 5.212, 3.14, etc. **Strings:** "Hello! I am a string!", "I'm also a string!" **Booleans:** true, false // just true or false! **Null/Undefined:** null, undefined // You'll see these a lot, basically if something doesn't exist and we try to access it, we'll get these back in return!

Numbers:

Open the console on codepen. In the console try writing some numbers. We can do any type of math with javascript that we could do with anything else! Here is some basic math:

```
4 + 20      // Addition
41.3 - 300    // Subtraction
10 / 3      // Division
15 * 3      // Multiplication
10 % 3      // Modulus (ONLY WORKS WITH WHOLE NUMBERS)
```

You're probably very familiar with all of these basic math operators, but you may not have seen modulus yet. Modulus is just a remainder, so in integer division, 10 divided by 3 gives us 3 remainder 1. When we use modulus, 10 % 3 = 1. 10 % 2 doesn't have any remainder, so we get: 10 % 2 = 0.

As a quick note, all of these operators still follow the order of operations*.

Strings:

```
"Hello World"
'Hello World'

//You can't do "Hello'
// Quotes must match!
// If you wanted to use an apostrophe, use double quotes, or you want to use double quotes in a string:
"I don't ever want to stop coding"
'He said "I like turtles" a lot'

//concatenation:
'Hello ' + 'World'
```

Writing out math and strings is great and all, but that's not what makes code so powerful. We need a way to save those numbers, letters, booleans, etc. That's where variables come in.

Variables:

Syntax:

var variableName = someValue const constantName = someUNCHANGINGValue

Take a look at the below code. Type the name of the variables in the console to see what they return:

```
// Numbers!
// Anything we could do with numbers we can also do with variables!
var a = 5
var b = 20
var c = a + b
var d = b / a

// Strings:
// Anything we could do with strings we can do with variables with strings in them!

var cityName = 'St. Louis '
var sportsTeam = 'Cardinals'

var sportsTeamInCity = cityName + sportsTeam


// You can do some neat things with strings:
```

```
var howLongIsSportsTeam = sportsTeam.length
var cityNameLength = cityName.length
```

A thing that variables can do is change! In javascript, they can change not only their value but also their type. Take a look:

```
var food = 'pizza'
food = 'cake'

food = 'the cake is a lie'

food = 5
```

Changing from string to number is completely fine in javascript! This can be extremely helpful, but it can also be very difficult if you're not paying attention (e.g. it would get really weird if you were calculating someone's bill on your site, and you tried to add 15 + 'Sweater').

EXERCISE:

In codepen, try these things:

1. Create two variables, one for you first name (called `firstName`) and one for your last name (called `lastName`).
    1. Store a string of your first name in `firstName`
    2. Store your last name in `lastName`
    3. Then create a new variable called `fullName`. Inside full name, add both first and last name together!
2. Create two constants (with `const` instead of `var`). Call them `a` and `b`. In `a` store **20**, and in `b` store **6**. In your codepen console, try the following:
    1. `a + b`
    2. `a - b`
    3. `a * b`
    4. `a / b`
    5. `a % b`

Built in JS Functions:

There are a number of functions we'll be using throughout the course, and most of those will be defined by us. There are a few functions that we'll want to know about before we start using a lot of javascript.

console.log():

To find out what's happening in our code, the very first function we'll look at is `console.log()`. This function takes whatever we pass into it, and logs it into the console (weird, huh?). Take a look:

```
console.log('Hello world')
var alsoHello = 'I also want to say hello'

console.log(alsoHello)

var mathStuff = 5 + 13

console.log(mathStuff)

console.log('You can concatenate too: ' + mathStuff + alsoHello)
```

Console.log is incredibly useful because it lets us print out a specific variable's data to the console so we can log what's happening in our website, or just for general debugging.

alert():

Arguably one of the most annoying functions, alert is a really helpful function because it quite literally alerts the user and won't let them do anything until they've acknowledged what the alert is:

```

var alertMessage = 'HEY USER! LOOK AT ME!'

alert(alertMessage)
```

This is often used for forgotten passwords or unsaved data, though it's not unheard of for sites to use alert for poor reasons.

prompt()

Prompt lets us very blatantly ask questions of our users. Just like alert, we can pass a string into our prompt, but this also comes with a text box for a user response:

```
    var nameQuestion = "What is your name?"
    prompt(nameQuestion)
```

You can then enter your name. However, did you notice that nothing happened with your name? Let's try saving that data:

```
    var nameQuestion = "What is your name?"
    var userName = prompt(nameQuestion)

    // get both alerts and console.logs!
    alert('Hi ' + userName)
    console.log('Hello! ' + userName)
```

EXERCISE:

In codepen, try these things:

1. Create a prompt to ask you your name. Store that name into a variable and then use it in an alert!
2. Take your code from earlier with your first name and your last name, and console log your full name!

Boolean logic:

Boolean logic is how we can do what is called "control flow". We'll get to how to control how our programs work a little later, but first, we need to know how to properly show off our boolean values! For now we'll start with just with boolean values with true and false:

AND:

```
For something to work with AND, both sides around the `&&` need to be true:
```

```
        true && false = false   // only one is true, so we ultimately get false
        false && true = false   // only one is true, so we ultimately get false
        false && false = false  // both are false! So naturally, false.
```

OR:

```
For something to work with OR, we just need one of the sides to be true around the `||` operator!
```

```
        true || false  = true   // one side is true, so we still get true
        false || true  = true   // one side is true, so we still get true, even though the first was false
        false || false = false  // both are false! So, we ultimately get false here.
```

```
The above are the basis of how we control the flow of our programs. This may not make sense now because well, true is true, that doesn't really tell us anything. Let's start looking at conditions:
```

Conditions:

```
When checking conditions, we can use basic greater than, less than, equal to, or not equal to in order to check what's going on with our program. These may not make the most sense at the moment (particuarly the checking for equality, but it'll make sense the more you use them!):
```

Less than / greater than:

1 < 5 is true because 1 is less than 5 5 < 2 is false because 5 is more than 2 2 > 5 is also false, because again, 5 is more than 2.

Equals / Not Equals:

Unlike less than and greater than, we need to do something different with checking for equality. This is because when we use 1 equals sign, in javascript that's the 'assignment' operator which we saw above in the variables section. To check for equality, we need to use `==`. Likewise, to check for things not being equal, we need to use `!=`:

```
1 == 1 is true because one does equal one.
1 == 5 is false because 5 and 1 are completely differet numbers.
1 != 5 is true because 1 is not equal to 5!
1 != 1 is false because 1 is equal to 1 (And by using the != operator, we're trying to see if they're not equal)
```

You can also check strings for equality!

```
"One String" == "Other String" is false because the strings don't say the same thing!
"One String" != "Other string" is true because the strings don't equal each other!
"Strings?" != "Strings?" is false because they are the same string, and we're checking for inequality!
```

Hopefully those make a little bit of sense, but what's great is that we don't have to use 'literals' (that is, non-variables) to check for some sort of boolean condition:

```
var oneIsLessThanFive = 1 < 5
console.log("it is " + oneIsLessThanFive + " that one is less than 5")

var oneIsGreaterThanFive = 1 > 5
console.log("it is " +oneIsGreaterThanFive + " that one is greater than 5")

var oneEqualsOne = 1 == 1
console.log("it is " + oneEqualsOne + " that one equals one!")

var fiveEqualsOne = 5 == 1
console.log("it is " + fiveEqualsOne + " that five equals one!")

var oneIsNotFive = 1 != 5
console.log("it is " + oneIsNotFive + " that one is not five")

var stringEquality = "Strings?" == "Strings?"
console.log("it is " + stringEquality + " that the two strings `Strings?` and `Strings?` are equal")

var stringNotEqual = "One String" == "Other String"
console.log("it is " + stringNotEqual + " that 'One string' is equal to 'Other string'")

var stringInequality = "One String" != "Other String"
console.log("it is " + stringInequality + " that 'One String' does not equal 'other'")

var sameStringInequality = "Strings?" != "Strings?"
console.log("it is " + sameStringInequality + " that 'Strings?' and 'Strings?' are not equal")
```

We can save the truthiness of those operators in variables like we just did, but what makes this REALLY powerful is that we can check the truthfulness of variables themselves! We'll take the exact same code as above, but now [everything will be a variable]!

```
var one = 1
var five = 5

var oneIsLessThanFive = one < five
console.log("it is " + oneIsLessThanFive + " that one is less than five")

var oneIsGreaterThanFive = one > five
console.log("it is " +oneIsGreaterThanFive + " that one is greater than five")

var oneEqualsOne = one == one
console.log("it is " + oneEqualsOne + " that one equals one!")

var fiveEqualsOne = five == one
console.log("it is " + fiveEqualsOne + "that five equals one!")

var oneIsNotFive = one != five
console.log("it is " + oneIsNotFive + " that one is not five")

var stringQuestion = "Strings?"
var oneString = "One String"
var otherString = "Other String"

var stringEquality = stringQuestion == stringQuestion
console.log("it is " + stringEquality + " that the two strings `Strings?` and `Strings?` are equal")

var stringNotEqual = oneString == otherString
console.log("it is " + stringNotEqual + " that 'One string' is equal to 'Other string'")

var stringInequality = oneString != otherString
console.log("it is " + stringInequality + " that 'One String' does not equal 'other'")

var sameStringInequality = stringQuestion != stringQuestion
console.log("it is " + sameStringInequality + " that 'Strings?' and 'Strings?' are not equal")
```

By doing what we just did above, we can now create really powerful programs!

It is also possible to [string the boolean logic together:]

```
var one = 1
var five = 5

var oneIsLessThanFive = one < five
console.log("it is " + oneIsLessThanFive + " that one is less than 5")

var oneIsGreaterThanFive = one > five
console.log("it is " +oneIsGreaterThanFive + " that one is greater than 5")

var eitherOR = oneIsLessThanFive || oneIsGreaterThanFive
var bothAnd = oneIsLessThanFive && oneIsGreaterThanFive

console.log("only one can be true, is it the 'eitherOr' or the 'bothAnd'? eitherOR: " + eitherOR + "     bothAnd:" + bothAnd)

// The above code for either or is the same as writing:
eitherOR = one < five || one > five
bothAnd = one < five && one > five

// either either or is true or both and is true, to say that we would write:
var ultimateOutcome = eitherOR || bothAnd


// the same way to write the above is:
ultimateOutcome = (one < five || one > five) || (one < five && one > five)

console.log('The ULTIMATE outcome: ' + ultimateOutcome)
```

Don't get too bogged down into trying to follow the logic. Just so long as you understand the basic's that's good enough (entire careers are spent in logic! We only need the basics).

Conditionals:

In the previous section, we talked about how you can check the truthiness of a statement and have it control the flow of your program. To do that, we use "if"s and "else"s.

Ifs Elses:

These follow the exact same logic as how we speak:

`If I eat icecream then I will be happy! ELSE (that is, if I don't eat ice cream) I will be very sad.`

What's different is that in the code, we need to have our sentence follow a special syntax:

```
if(some truthy statement) {
  // MAKE SURE TO WRAP THIS IN CURLY BRACES LIKE SO:
  console.log('someSpecialOutcome is happening because our truthy statement was true')
} else {
  console.log('some other outcome is happening if the trythy statement is false')
}
```

Let's code this out:

```
var snackIWillEat = 'iceCream'

if(snackIWillEat == 'iceCream'){
  alert('I AM VERY HAPPY!')
}
else {
  console.log('NOW I AM SAD')
}
```

This code alerted us that I was happy because I ate ice cream! What if I didn't, though? Let's try it again:

```
var snackIWillEat = 'broccoli'
if(snackIWillEat == 'iceCream'){
  alert('I AM VERY HAPPY!')
}
else {
  console.log('Well now I'm sad')
}
```

Now we see that in the console, I printed that I was sad. That's because 'broccoli' just isn't ice cream.

It's fun to know whether or not someone will be sad when they don't eat ice cream, BUT why are conditionals useful? What if you needed somebody to log into a website? You would need a conditional to verify that they were who they said they were!

```
var userName = prompt("Who are you?")

if(userName == "Spice Girls") {
  alert("SPICE UP YOUR LIFE")
}
else {
  alert("Get out of here you unspicy person!")
}
```

Now, let's take a second and talk about using multiple ifs! [You can "nest" if/else statments:]

```
var userName = prompt("Who are you?")

if(userName == "Spice Girls") {
  alert("SPICE UP YOUR LIFE")
} else if (userName == "Beastie Boys"){
  alert("FIGHT FOR YOUR RIGHT TO PARTY!!")
}
else {
  alert("Get out of here you unspicy person!")
}
```

You can nest as many if elses as you want! If this looks super weird, don't worry too much on syntax now. We'll be doing this a lot, and it will eventually sink in!

EXERCISE:

In codepen, try these things:

1. Prompt the user twice and ask for two numbers. Store those numbers into two variables `a` and `b`. Add them together and store them in a third variable called `c`. Alert the user what the new var `c` is holding!

Loops:

Loops are a way for us to do a bunch of things without writing a lot of code! Suppose I wanted to, for some crazy reason, print out "I LIKE TURTLES" 100 times. [You could write]:

```
console.log("I LIKE TURTLES")
console.log("I LIKE TURTLES")
console.log("I LIKE TURTLES")
console.log("I LIKE TURTLES")
console.log("I LIKE TURTLES")
console.log("I LIKE TURTLES")
...
... // Assume we wrote 100 console logs
...
console.log("I LIKE TURTLES")
console.log("I LIKE TURTLES")
console.log("I LIKE TURTLES")
console.log("I LIKE TURTLES")
```

We could write 100 console logs. That's crazy though. We're very busy people and we don't have time to do that! Instead we can use a loop! The two loops we'll be learning about are while loops and for loops!

while:

A while loop is very simple it looks like:

```
while( SOME TRUTHY STATEMENT ) {
  // DO STUFF HERE
}
```

So to write "I LIKE TURTLES" 100 times to the console, you simply write:

```
var count = 0

while ( count < 100) {
  console.log("I LIKE TURTLES")
  count = count + 1      // this increments the count! I looks a little weird, but it's basically setting count equal to its old value + 1.
  }
```

Note: If you don't have a conditional statement in the the parentheses of the while loop, it will run forever (sometimes you'll want that, but most of the time you wont)!:

```
var count = 0

while () {
  console.log("I LIKE TURTLES: " + count)
  count = count + 1      // this increments the count! I looks a little weird, but it's basically setting count equal to its old value + 1.
}
```

You might be wonder what the point of a loop is other than printing "I like turtles" a bunch of times. Suppose you have a login. What if your user enters the wrong information on accident? Let's take the nested if else from above and use that:

```
var count = 0

while(count < 5){
    console.log('count', count)

  var userName = prompt("Who are you?")

  console.log("Your name is: ", userName)

  if(userName == "Spice Girls") {
    alert("SPICE UP YOUR LIFE")
  }
  else if (userName == "Beastie Boys"){
    alert("FIGHT FOR YOUR RIGHT TO PARTY!!")
  }
  else {
    alert("Get out of here you unspicy person!")
  }

  count = count + 1
}

console.log("it's over!")

```

EXERCISE:

Write a while loop that counts to 10 and prints it to the console!

For loops:

For loops are the exact same as while loops, but they're a bit more compact! They look like:

```
for( initializer ; condition to check ; something to do after the code runs ){
  // looped code goes here
}
```

What that looks like is:

```
for (var i = 0; i < 10; i = i + 1){
  console.log("I like to repeat myself. " + i + " times")
}
```

You may be wondering what the point of a for loop is, since we can do everything in a while loop just as easily. A lot of people use for loops so that they can easily iterate over arrays (which we're talking about next)!

EXERCISE:

Take your while loop from before, and convert it to a for loop!