**num += 1**; this is the same as typing **num = num + 1**. You may sometimes see **num++** or even **++num**; both are equivalent to **+=**.

Math Library

Math.floor() makes negative numbers more negative and positive less positive. Conversely, Math.ceil() makes positives more positive and negatives less negative. Math.trunc() drops any fraction, moving the number towards zero.

Math.floor(2.718) and Math.trunc(2.718) both return 2, but Math.ceil(2.718) returns 3.

Math.floor(-3.1416) is -4; both Math.trunc(-3.1416) and Math.ceil(-3.1416) are -3.

Naturally, Math.ceil(42) == Math.trunc(42) == Math.floor(42) == 42

What if you want a random integer as low as 51 and as high as 100? Math.random() is "from 0 to almost-one". Math.random() \* 50, then, is "from 0 to almost 50". Let's turn those decimal ranges into integers: Math.trunc(Math.random() \* 50) is "50 possible integers from 0 to 49". Let's add an offset, so we start at 51: Math.trunc(Math.random() \* 50)+51 is perfect. Whew!

**Using Modulo to Extract a Digit**

If variable myBigNum contained a big number, how would you get the value of the ‘hundreds’ digit? (To review, the *hundreds* digit of 32768 is ‘7’, since it is thirty-two thousand, seven hundred sixty-eight.) First, you might myBigNum = myBigNum / 100, to shift the decimal point to exactly where you want (result: 327.68). Then, Math.floor(myBigNum) could remove any decimal leftovers (result: 327). Finally, you could use % to extract only the ‘ones’ digit. *What is % and how does it work?*

The % (modulo) operator often goes with / (divide). Basically, % *divides the first number evenly by second and returns the remainder*. Example: 255 % 20 is 15, and 16 % 2 is 0. And … 327 % 10 becomes the 7 we want. Putting it all together, we print the ‘hundreds’ digit like this:

console.log(Math.floor(myBigNum / 100) % 10);

Let’s decode this. Formulas are computed inside out; so start with the division inside the parentheses. Once we’ve done that, our formula is simplified:

console.log(Math.floor(327.68) % 10);

Then comes the floor call. After evaluating that, our formula is:

console.log(327 % 10);

And finally, comes the modulo operation, after which we have this!

console.log(7);

Objects

console.log(Dany.last\_name); // Targaryen

But with arrays, we cannot find a named index. To accomplish that, we need to use a structure called an object. An object allows us to find information inside of a collection by naming a key to that value. Let's take a look at that in action.

var Dany = {

first\_name: 'Daenerys',

last\_name: 'Targaryen',

age: 22,

allegiance: 'House Targaryen',

ancestry: 'Valyrian',

titles: ['Queen of the Andals, the Rhoynar and the First Men', 'Lady Regent of the Seven Kingdoms', 'Protector of the Realm', 'Khaleesi of the Great Grass Sea', 'Breaker of Chains', 'Mother of Dragons', 'Queen of Meereen']

}

When creating an Object, we might not create it correctly the first time. Let's make another Object that represents a Dojo:

var dojo = {}; // creates an empty object

dojo = {

name: 'Coding Dojo', // property can store a string

number\_of\_students: 25, // property can store a number

instructors: ['Andrew', 'Michael', 'Jay'], // property can store arrays

location: { // property can even store another object!

city: 'San Jose',

state: 'CA',

zipcode: 95112

}

} // access object properties with dot (.) notation

console.log(dojo.name, dojo.number\_of\_students, dojo.instructors, dojo.location);

console.log(dojo["name"]); // or bracket [] notation (note: specify key in quotes)

# Arrays of Objects

var glazedDonuts = [

{

frosting: 'glazed',

style: 'cake',

type: 'old fashioned',

age: '45',

time: 'minutes'

},

{

frosting: 'glazed',

style: 'yeast raised',

type: 'regular',

age: '5',

time: 'minutes'

},

{

frosting: 'glazed',

style: 'yeast raised',

type: 'jelly filled',

age: '1',

time: 'seconds'

}

];

You could then go to the donut owner and ask something like: Can I buy the 1st donut on the rack if it has been out of the oven for fewer than 25 minutes? The code conversation for that would be:

var purchase;

//You

if((glazedDonuts[0].age < 25 && glazedDonuts[0].time == 'minutes') || glazedDonuts[0].time == 'seconds'){

//shop owner

purchase = glazedDonuts[0];

}

else {

console.log('sorry it has been out a bit longer');

}

#### Quiz

1. What would $('div p') select in Jquery?
   *   all paragraphs within a div
   *   all div and all paragraphs
   *   nothing
   *   all divs within a paragraph
2. How do we access jQuery?
   *   importing their code by adding a 'src' attribute to our JavaScript tags, either a local version or remote
3. What is jQuery?
   *   A javascript library
4. Which sign does jQuery use as a shortcut for jQuery?
   *   the $ sign
5. With jQuery, look at the following selector: $('p'). What does it select?
   *   all of the paragraphs
6. What is the correct jQuery code to set the background color of all p elements to red
7. With jQuery, look at the following selector: $('div.intro'). What does it select?
   *   All div elements with class='intro'
8. What would the following do? $('p, a, #dojo').addClass('red')
   *   add class 'red' to all paragraphs, all links and an element that has an id dojo
9. How is $('p.dojo') different from $('p .dojo')?
   *   $('p.dojo') is selecting all PARAGRAPHS with the class name of 'dojo' while $('p .dojo') is referring to elements with class 'dojo' that are INSIDE the paragraphs
10. Can paragraphs (p tag), links (a tag), input fields (input tag) all have their own id and class?
    *   Of course! You can give an id or a class to any html tag (p, a, body, div, select, input, span, etc)
11. How do you give multiple classes to an element?
    *   You could do it by doing things like < a href='http://www.codingdojo.com' class='dojo black awesome' >anchor text < /a >. This code would add three classes to the a tag: dojo, black, and awesome.
12. What does console.log() do?
    *   Displays any expression the user inputs into the JavaScript console of your browser.
13. What does the selector $(this) allow us to do?
    *   Select the particular element that triggered the event handler, without having to refer to it by ID
14. What is the difference between .html() and .append()??
    *   .html() overwrites the markup of an element, .append() adds to the markup

A **callback** is nothing more than calling a function at some time from within another function