JS Notes

Type Coercion

Process of converting a value from one type to another

```
7 + 7 + 7 = 21
7 + 7 + "7" = "147"
"7" + 7 + "7" = "777"
```

After first string value, all values are converted to strings and concatenated.

```
1 == true // true
7 == "7" // true
1 === true // false*
7 === "7" // false*
```

* *Prevents type coercion*

JS Functions

Functions inherit from object and may be passed or stored like any variable

A function tied to an object is a method

No function overloading; missing parameters are set to undefined and extras are ignored

The "this" parameter – every function has on, what it refers to is bound at invocation

Invocation forms – how can we call functions?

Function Form:

```
doThings(args);
```

"this" refers to global object

Method Form:

```
obj.doThings(args); // OR
obj.[doThings][args]
```

"this" refers to the object to which the form belongs

Constructor Form:

```
new someObj(args);

var someObj = function(name, age) {
      this.name = name;
      this.age = age;
}
```

"this" refers to new object being created (this is best practice)

You should also capitalize constructors

Apply Form:

```
doThings.apply(someObj, [args])

var me = new someObj(…);
```

invoking the method while dynamically setting the object

"this" refers to `someObj`, which is passed into the apply method

Scopes

Function: When declared inside of a function, accessible only within that function

Block: Introduced with ES6; only available via use of "let" or "const" instead of var

Global: Declared outside any function or never declared at all (this is also called an implied global). Accessible everywhere; potential for renaming variables, but difficult for debugging and testing code

JS *hoists* variable declarations to the top of their respective scopes IF declared. If not, variable declarations are always hoisted to global scope

JSON

JavaScript Object Notation

Please note, JSON is not the same as a JS object!

It's a language agnostic way to represent data that was inspired by, but not the same as the way JavaScript represents objects

```
{
"key" : value,
"key" : value,
"key", {
      "key" : value,
      "key" : value
      }
}
```

Nesting is OK

Must use double quotation marks (""), single quotes not allowed (')

Keys and string values must be in quotes

Values may be strings, numbers, JSON objects, arrays, true/false, and null (basically, anything a JS object can hold EXCEPT functions and undefined)