**Chapter 2 Programming basics:**

Commenting is the same as C++

Semicolon not required but can save headaches later - USE THEM

**Reserved Words**:

* Abstract
* Awit
* Boolean
* Break
* Byte
* Case
* Catch
* Char
* Class

**Not Reserved** (but treat them like they are)

* Undefined
* NaN
* Infinity

**Primitive Data Types** (everything not listed is an object)

* String
* Symbol
* Number
* Boolean
* Undefined
* Null

**Variables** declared with ‘let’ or ‘const’

**Naming** – Follow the same rules used in C++ and you’re safe

**Wrapper Objects**: allows you to treat primitives like objects

**Template Literals**: use the backtick character, ` , to delaminate the string.

This allows both types of quotes to be used in string

Also, allows JavaScript expression to be inserted inside a string and the results will be displayed

They can contain line breaks which get converted to \n

If you want to use a backtick in a template literal, it needs to be escaped

**Symbols**: the only primitives that DON’T have a literal form, the only way to create them is to use the Symbol() function

You need to add a description of the symbol inside the parentheses

The main use for symbols is as object property keys

**Numbers:**

If a number starts with 0x, it is considered to be hexadecimal

If a number starts with 0o, it is considered to be octal

**Chapter 3 Arrays, Logic and Loops**

Removing an item from an array replaces it with *undefined*, but the location still exists. This means that the array still has the same number of elements

**Destructing Arrays:** is the concept of taking values out of an array and presenting them as individual values. Even though assignment is made using an array, each variable exists on its own outside the array.

Array *length* property is mutable, meaning you can manually change it. If you make the array longer, the extra space will be filled in with *undefined*. If you make it shorter, the extra elements will be removed completely.

**Pop, Push, Shift, Unshift**

pop, removes the last element of the array

shift, removes the first item in the array

push, appends a new value to the end of the array

unshift, appends to the beginning of the array

**Merge Arrays** using concat() method

**Join() method** turns an array into a string that comprises all the items in the array, separated by commas

**Slice()** creates a sub array (doesn’t actually remove elements from the array)

**Splice()** removes items from the array and replaces them. It can also be use to insert values into the array starting at a specific index, without removing data that was already in the array

You can use splice() to remove an item from an array by making the length of the splice 1 and not adding a new value

**Reverse(), Sort(), indexOf()**

**Multi-dimensional Arrays**: Remember this from C++

**Spread Operator \*\*\*\*\*** look into this more to understand it

**Sets** is a data structure that represents a collection of unique values, so it cannot include and duplicate values

\*Sets cannot contain duplicate values

You can add multiple values to a set by putting them in an array, but if the same value appears in the array more than once, it will only be put into the set one time.

All non-primitive values such as arrays and objects are considered unique, even if the values inside each array are the same.

The string ‘2’ and the number 2 can both be in a set

\*A set can be converted to an array (pg. 137)

WeakSet() --- read more about this, maybe ask questions

**Ternary operator** is shorthand for writing an if…else statement like this

Condition ? (//code to run if condition is true) : (//code to run if condition is false)

\*\*The **for-of** loop is interesting DON’T FORGET ABOUT IT.

**Chapter 4 Functions**

Functions are considered to be “first-class” objects.

\* This means they behave like any other object, meaning they can be assigned to a variable, stored in arrays and can even be returned by another function.

**Function Declaration**

Function hello() {

Console.log(‘Hello World!’);

}

**Function Expression**

const goodbye = function() {

console.log(‘Goodbye World!’);

}; 🡨 note that a function expression ends with a semicolon

\*This is an anonymous function because it doesn’t have a name

const goodbye = function bye() {

console.log(‘Goodbye World!’);

}; 🡨 note that a function expression ends with a semicolon

\*This is a named function expression because it has a name

\*\* It is not recommended that you use a constructor function (that is why I am not putting an example)

**Arrow Functions**

A syntax that makes declaring functions much more succinct by using less verbose syntax.

Arrow functions are always anonymous. (parameter(s) comes before the arrow)

const square = x => x\*x;

\*Multiple parameters must be in parenthesis:

const square = (x, y) => x + y;

\*If there are no parameters, and empty set of parentheses are required before the arrow:

const square = () => alert(‘Hello World’);

**Advantages**:

* Less verbose
* Single parameters don’t need to be in parenthesis
* Body doesn’t need to be inside a block if it is only one line
* return keyword isn’t required if the return statement is the only statement in the body
* Doesn’t bind its own value of this to the function

**Hoisting:**

This is complicated…

Functions are hoisted; therefore, they don’t need to be declared before they are invoked.

Functions expressions don’t get hoisted, and must be declared before being invoked.

Variables declared with const or let do not get hoisted

Variables declared with var do (I think. At least sometimes) ***…Need more clarification on this***

A **Callback** is a function being given to another function as a parameter.

**Reduce(), Filter(),**

**Chaining iterators** this is confusing to me… probably need to understand this much better.