Week 2 Notes

**CH 2 Programming Basics**

Reserved variable names in JavaScript:

abstract, await, boolean, break, byte, case, catch, char, class, const, continue, debugger, default, delete, do, double, else, enum, export, extends, false, final, finally, float, for, function, goto, if, implements, import, in instanceof, int, interface, let, long, native, new, null, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, transient, true, try, typeof, var, volatile, void, while, with, yield

Assignment by reference:

const c = { value: 1 };

let d = c; // c.value = 1, d.value = 1

d.value = 2; // c.value = 2, d.value = 2

Backticks allow for JavaScript to be placed in print statements

const age = 39;

`I will be ${ age + 1 } next year`;

<< 'I will be 40 next year'

NaN:

Occurs when an operation is attempted and the result isn’t numerical. Ex. String \* a number

Type Coercion can be tricky:

'2' \* 8;

<< 16

'2' + 8;

<< '28'

In the last problem, ‘2’ must be specified as an int before adding.

Ex. Number(‘2’) + 3 = 5; (you may also use parseInt)

To convert a number to a string:

String(3) = ‘3’

Null & Undefined:

 Null means 'no value'. It can be thought of as a placeholder that JavaScript uses to say "there should be a value here, but there isn’t at the moment."

If this reminds you a lot of undefined then this is because they are both 'non-value' values, meaning they are similar, but behave slightly differently. For example, if you try to do sums with them:

10 + null; // null behaves like zero

<< 10

10 + undefined; // undefined is not a number

<< NaN

 null is coerced to be 0 , making the sum possible whereas undefined is coerced to NaN , making the sum impossible to perform.

In general, values tend to be set to undefined by JavaScript, whereas values are usually set to null manually by the programmer.

**CH 3 Arrays, Logic, and Loops**

Removing values from an array:

Delete avengers[3]; from the list [‘Cap’, ‘Iron man’, ‘Thor’, ‘Hulk’]

Would delete Hulk and replace it with undefined [‘Cap’, ‘Iron man’, ‘Thor’, ‘undefined’]

Swapping in JavaScript:

[x,y] = [y,x];

Pop, Push, Shift, Unshift:

Will not leave an undefined value

Maps and sets must be created by:

Const name = new Map() or new Set()

Weak maps and sets will enable the garbage collector to automatically remove any dead entries (preventing memory leaks)

Looping over Maps:

const romanNumerals = new Map();

romanNumerals.set(1,'I').set(2,'II').set(3,'III').set(4,'IV').set(5,'V');

romanNumerals

<< Map { 1 => 'I', 2 => 'II', 3 => 'III', 4 => 'IV', 5 => 'V' }

Every map object has a keys() method lets us iterate over each key with the following for-of loop:

for(const key of romanNumerals.keys()) {

console.log(key);

}

<< 1

2

3

4

5

There is also a values() method that lets us iterate over the values in a similar way:

for(const value of RomanNumerals.values()) {

console.log(value);

}

<< I

II

III

IV

V

**CH 4 Functions**

Defining a function:

Function Declarations:

function hello(){

console.log('Hello World!');

}

Function Expressions: (anonymous functions) (make sure to finish expression with semi-colon)

const goodbye = function(){

console.log('Goodbye World!');

};

Arrow Functions:

Benefits of arrow functions:

* They are much less verbose than normal function declarations.
* Single parameters don't need putting into parentheses.
* The body of the function doesn't need placing inside a block if it's only one line.
* The return keyword isn't required if the return statement is the only statement in the body of the function.
* They don't bind their own value of this to the function (we'll see why this is a particularly useful property when we cover objects later in the book).

Always anonymous functions

const square = x => x\*x;

If no parameters needed, empty parentheses must be placed before arrow:

Const hello = () => alert(‘Hello World’);

Callback: (can be used to call the function…back)

function sing(song,callback) {

console.log(`I'm singing along to ${song}.`);

callback();

}