Week 2 Notes- Programming basics

Comment in your code!!!

// This is an example of a short comment

/* This is an example of a longer,

Multi-line comment. Put notes.

*/

Javascript is a C-style syntax because similar with the C programming language.

Best Practice – Write each statement on new line, terminated by semi-colon.

```
const message = 'Hello World!';
alert(message);
```

A Block: Statements that are collected inside curly braces:

```
{
// this is a block containing 2 statements
    const message = 'Hello!';
    alert(message);
}
```

Use whitespace (spaces, tabs, new lines) to format code to be readable

Reserved words:

Don’t use these to name variables or function parameters etc.:

```
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
```

Also undefined, NaN, Infinity

Data Types:

- String
- Symbol (The symbol primitive data type was only introduced in ES6.)
- Number
- Boolean
- Undefined
- Null

Any value not a Data type above is an OBJECT. They include arrays, functions, and object literals.

OPERATOR applies an operation to a value, known as the OPERAND

Unary and Binary operators:

typeof ‘hello’

Operator: typeof (unary – needs one operand) Operand: ‘hello’

3+5

Operator: +(binary – needs two operands) Operands: 3, 5

Also ternary operator

VARIABLES – refer to a value stored in memory

Declare and assign:

const and let

const: variable will not be reassigned to another value

let: variable might be reassigned later

Can declare and assign multiple variables in same line:

let x=3, y=4, z=5;

const will not make object, function or array immutable:

```
const name = { value: 'Alexa' }; // an object
name.value = 'Siri'; // change the value
<< 'Siri'
```

*Use const to declare most variables – helps avoid bugs

var is older way of declaring – doesn’t prevent you from overwriting things – use let

Scope:

const and let = block scoped(value only exists inside the black they are declared in)

Global scope: Variable outside of a block – accessible everywhere

*Keep number of global variables to a minimum! May clash/override each other

Local scope: Blocks are used to create a local scope. Variable is only visible in the block.

*Use const or let in block or will use global variable if available – GOOD PRACTICE to always use them to declare variables.

Naming constants and variables:

Name them something that describes what it represents

Can start with any uppor or lower case letterm underscore, or $. Can contain numbers, but not start with them.

Variable starting with underscore refer to private properties and methods.

Variable starting with $ used by jQuery

Case senditinve

Best convention for writing more than one word: camel-case (ex. firstNameAndLastName)

Direct assignment and by reference:

Direct:

```
const a = 1;
let b = a; // a = 1, b = 1
b = 2; // a = 1, b = 2
```

Reference:

```
const c = { value: 1 };
let d = c; // c.value = 1, d.value = 1

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

Strings:

Constructor function:

new String(“hello”)
<< [String: ‘hello’]

Creates new string same as string literal ‘hello’ but classed as an object. Preferable to use string literal.

Access properties of string using dot notation:

const name = ‘Alexa’;

name.length; //answer is 5

Can also use square brackets:

name[‘length’]; //answer is 5

call a method using dot notation:

name.toUpperCase(); //’ALEXA’

name.toLowerCase(); //’alexa’

Concatenate strings using “+”:

‘Java’ + ‘Script’ + ‘ ‘ + ‘Ninja’; //JavaScript Ninja

Use backtick ` for template literals:

Able to use both types of quote marks in a string: `She said, “It’s Me!”`

Also allow INTERPOLATION:

```
        const name = `Siri`;
`Hello ${ name }!`;
<< 'Hello Siri!'
const age = 39;
`I will be ${ age + 1 } next year`;
<< 'I will be 40 next year'
```

Symbols:

Create unique values to avoid naming collisions. Use the Symbol() function:

const uniqueID = Symbol();

Recommended to add description in parentheses for debugging in the log:

const uniqueID = Symbol('this is a unique ID');

Also manually access usning the String() function:

String(uniqueID) // 'Symbol(this is a unique ID)'

It is a primitive data type – typeof should bring back 'symbol'

Numbers:

Integers or floats – Both are given the type of 'number' -Number.isInteger() used to check if number is an integer:

Number.isInteger(42); //True

Number.isInteger(3.14); //False

Constructior function for numbers: new Number(3) //[Number: 3]

Exponential notation: 1e6; //means 1 multiplied by 10 to the power 6 (1000000), 2e3; //2 X 10^3 (2000)

Method toFixed() (round a number to a fixed number of decimals)

Arithmatic operations:

Addition +, Subtraction -, Multiplication *, Division /, Exponentiation **,
remainder of division (modulus) %

Compound assignment operator +=:

Ex. let points = 0;

points = points + 10; or

points += 10;

also:

points -= 5; //decreases points by 5
points *= 2 // doubles points
points /=3; //divides vale of points by 3
points %=7; //changes the value of points to the remainder if its current value is divided by 7

Incrementing values:

++ to increment a value by 1.

let points = 5;
points ++

points++; //will return original value then increase by 1
++points; //will increase value by one then return the new value

-- decrease by 1

Infinity:

Error value used to represent a number that is too big for the language – can only handle 1e308

Also caused when dividing by 0

NaN:

"Not a Number" – operation attempted and result isn't numerical – ex. multiply string by a number

Can use Number.isFinite() method to check if a value isn't Infinity, -Infinity, or NaN

Correctly convert strings to numbers: Number() method. Number('23'); //23

Convert numbers to strings: String() method. String(23); //'23' or toString() method 10.toString(); //'10'

Parsing Numbers:

parseInt() method used to convert a string representation of a numerical value back into a number.

Undefined: variable has not been assigned a value.

Null: No value

Logical Operators:

! (not)

!! (double negation)

&& (logical AND) Must satisfy both conditions

|| (logical OR) Must satisfy at least one condition

Comparison Operators:

=

== (Soft Equality) can be different data types ex. 5 and '5'

=== (Hard Equality) true if same data types – Always use hard equality when testing if two values are equal. Avoid type coercion problems

!= (Soft Inequality)

!== (hard inequality)

>

<

>=

<=

Arrays, Logic, and Loops

Arrays:

ARRAY: ordered list of values. An object.

Array literal: const myArray = []; //better to stick with this
Array constructor function: const myArray = new Array();
Find value of element at 0: myArray[0]; //undefined because it is an empty array
Add a value to an array: myArray[0] = 'Superman'; myArray[3] = 'Batman';

View array by typing the name into the console:
myArray; //['Superman', undefined, undefined, 'Batman']

Remove values: delete myArray[3];

myArray; //['Superman', undefined, undefined, undefined]

Deleted 'Batman' but now myArray[3] is undefined.

Destructing Arrays: Taking values out of an array and presenting as individual values.

Const [x, y] = [1 2]; //x=1 y=2

```
const avengers = ['Captain America', 'Iron Man', 'Thor', 'Hulk',
'Hawkeye', 'Black Widow'];
```

avengers.length; //6

Find last item in array: avengers[avengers.length-1]; //'Black Widow'

Can change the length: avengers.length = 8;

avengers; //['Captian America', 'Iron Man', 'Thor', 'Hulk', 'Hawkeye', 'Black Widow', undefined, undefined]

Pop, Push, Shift, Unshift:

avengers.pop(); //remove last item from array

avengers.shift(); //remove first item in array

avengers.push('Thor'); //Append item to end of array

avengers.unshift('Captian America'); //Add item to start of array

Merging arrays

concat() method:

avengers.concat(['Hulk', 'Hawkeye', 'Black Widow']);

*does not change the array, but makes a new array that combines the two. Have to use assignment to update the array to the new one:

avengers = avengers.concat(['Hulk', 'Hawkeye', 'Black Widow']);

Spread Operator: avengers = [...avengers, ... ['Hulk', 'Hawkeye', 'Black Widow'];

Join() method: turn the array into a string with all of the items in the array separated by commas:

avengers.join(); // 'Captain America, Iron Man, Hulk, Hawkeye, Black Widow'

avengers.join('&'); //'Captain America & Iron Man & Thor & Hulk & Hawkeye & Black Widow'

slice(): Creates a subarray and is non-destructive

splice(): Removes items from an array then inserts new items in their place and is destructive

reverse(): reverse the order of an array

sort(): Sort the order of an array. (Alphabetically, also numbers by first number.

Is a value in an array? indexOf(): find the first occurrence of a value in an array. Returns index. If not in array, returns -1.

```
        Multidemensional array: an array of arrays. const coordinates =
[[1,3],[4,2]];
<< [[1,3],[4,2]]
```

To access the values in a multidimensional array, we use two indices: one to refer to the item's place in the outer array, and one to refer to its place in the inner array:

```
coordinates[0][0]; // The first value of the first array
<< 1
coordinates[1][0]; // The first value of the second array
<< 4
coordinates[0][1]; // The second value of the first array
<< 3
coordinates[1][1]; // The second value of the second array
<< 2
```

The spread operator that we met earlier can be used to *flatten* multi-dimensional arrays. Flattening an array involves removing all nested arrays so all the values are on the same level in the array. You can see an example of a flattened array below:

```
const summer = ['Jun', 'Jul', 'Aug'];
const winter = ['Dec', 'Jan', 'Feb'];
const nested = [ summer, winter ];
<< [ [ 'Jun', 'Jul', 'Aug' ], [ 'Dec', 'Jan', 'Feb' ] ]
const flat = [...summer, ...winter];
<< [ 'Jun', 'Jul', 'Aug', 'Dec', 'Jan', 'Feb' ]
```

Convert sets to arrays:

A set can be converted into an array by placing the set, along with thespread operatordirectly inside an array literal.

To demonstrate this, first we'll create a set of three items:

```
const shoppingSet = new Set().add('Apples').add('Bananas').add('Beans');
shoppingSet
```

```
<< Set { 'Apples', 'Bananas', 'Beans' }
```

Then we convert it into an array:

```
const shoppingArray = [...shoppingSet]
shoppingArray
<< [ 'Apples', 'Bananas', 'Beans' ]
```

It's also possible to use the `Array.from()` method to convert a set into an array. The following code would achieve the same result as using the spread operator above:

```
const shoppingSet = new Set().add('Apples').add('Bananas').add('Beans');
const shoppingArray = Array.from(shoppingSet);
```

By combining this use of the spread operator with the ability to pass an array to the `new Set()` constructor, we now have a convenient way to create a copy of an array with any duplicate values removed:

```
const duplicate = [3, 1, 4, 1, 5, 9, 2, 6 ,5,3,5,9];
<< [ 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 9 ]
const nonDuplicate = [...new Set(repeatedArray)];
<< [ 3, 1, 4, 5, 9, 2, 6 ]
```

Maps: List of Key Value Pairs

Logic:

If Statements:

Else statements:

Ternary Operator: a shorthand way or writing an if...else statement (?)

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

Switch statements:

```
                switch (number) {
    case 4:
    console.log('You rolled a four');
    break;
    case 5:
```

```
    console.log('You rolled a five');
    break;
    case 6:
    console.log('You rolled a six');
    break;
    default:
    console.log('You rolled a number less than four');
    break;
}
```

Loops:

While Loop:

Infinite Loops:

Do… while loops:

For loops:

Nested for loops:

Looping over arrays:

```
  for(let i=0, max=avengers.length; i < max; i++){
console.log(avengers[i]);
}
```

Looping over sets:

Looping over maps:

Functions:

Function Declaration:

Function hello(){

console.log(‘Hello World!’);

}

Function Expression: assigns an anonymous function to a variable

const goodbye = function(){

console.log(‘Goodbye World!’);

};

Every function has a name property

Invoke a function: need the parentheses to invoke or else you are just referencing the function.

```
hello();
```

Keep code DRY- Don't repeat yourself.

Return values: value that comes after return keyword. If not specified, a function will return undefined.

```
function howdy(){
    return 'Howdy World!';
}
const message = howdy();
    //'Howdy World!'
```

Parameters and Arguments:

Often used interchangeably for input values provided for the function.

```
function square(x){
    return x*x;
}
square(4.5);
//20.25
```

Multiple parameters:

```
function mean(a,b,c){
    return (a+b+c)/3;
}
mean(1, 3, 6);
    //3.33333333335
```

Variable number of Arguments:

Every function has a special variable call arguments. It is an array-like object that contains every argument passed to the function when it is invoked. You can access using index notation: arguments[0].

arguments is not an array. It has length property, can read and write each element with index notation, but no array methods.

Use the rest operator. Used to deal with multiple arguments by creating an array of arguments that are available inside the body of the function.

function rest(…args){

return args;

}

The args parameter is an actual array and has access to the array methods.

function rest(…args){

for(arg of args){

console.log(arg);

}

}

rest(2,4,6,8);

//2
4
6
8

Improved mean function using rest parameter:

```
function mean(...values) {
    let total = 0;
    for(const value of values) {
        total += value;
    }
    return total/values.length;
}
```

Default parameters:

```
function hello(name='World') {
    console.log(`Hello ${name}!`);
}
hello();
<< 'Hello World!'
hello('Universe');
<< 'Hello Universe!'
```

Use Default parameters after non default parameters!

Arrow functions:

Parameters come before the arrow and the main body of the function comes after.

Const square = x => x*x;

Const add = (x,y) => x+y;

No parameters:

Const hello = () => alert('Hello World!);

Longer functions still require curly braces for the body of the function

Function Hoisting:

Moving all variable and function declarations to the top of the current scope, regardless of where they are defined.

```
// function is invoked at the start of the code
            hoist();
// ...
// ... lots more code here
// ...
// function definition is at the end of the code
function hoist(){
    console.log('Hoist Me!');
}
```

Variable Hoisting:

Var keywords are automatically moved to the top of the current scope. Variable assigned at end of a function will have value of undefined until the assignment is made.

```
console.log(name); // will return undefined before assignment
// variable is defined here
var name = 'Alexa';
console.log(name); // will return 'Alexa' after assignment
```

Callbacks:

Function can be given as a parameter to another function

```
        function sing(song) {
    console.log(`I'm singing along to ${song}`);
}
sing('Let It Go')
<< 'I'm singing along to Let It Go'
```

We can make the `sing()` function more flexible by adding a `callback` parameter:

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

```
    callback();
}
```

```
function dance() {
    console.log("I'm moving my body to the groove."); ( We're just logging a simple message to the console in these examples, but these functions could be used to do anything in a practical sense.)
}
```

Now we can call our `sing` function, but we can also dance as well as sing:

```
sing('Let It Go',dance);
<< 'I'm singing along to Let It Go.'
'I'm moving my body to the groove.'
```

You can sort arrays of numbers with a callback:

provide a callback function to the `sort()` method that tells it how to compare two values, `a` and `b` . The callback function should return the following:

- A negative value if `a` comes before `b`
- `0` if `a` and `b` are equal
- A positive value if `a` comes after `b`

Here's an example of a `numerically` function that can be used as a callback to sort numbers:

```
function numerically(a,b){
    return a-b;
}
```

This simply subtracts the two numbers that are being compared, giving a result that is either negative (if `b` is bigger than `a`), zero (if `a` and `b` are the same value), or positive (if `a` is bigger than `b`).

This function can now be used as a callback in the `sort()` method to sort the array of numbers correctly:

```
> [1,3,12,5,23,18,7].sort(numerically);
<< [1, 3, 5, 7, 12, 18, 23]
```

Array iterators:

forEach() will loop through the array and invoke a callback function using each value as an argument.

map() iterates over an array and takes a callback function as a parameter that is invoked on each item in the array.

reduce() iterates over each value in the array, but it cumulatively combines each result to return just a single value.

filter() returns a new array that only contains items from the original array that return true when passed to the callback.

You can chain iterators together:

```
        const sales = [ 100, 230, 55];
totalAfterTaxSales = sales.map( (amount) => amount * 1.15 ).reduce( (acc,val) => acc
+ val );
<< 442.75
```