Week II Section Readings

Chapter I – Hello JavaScript.

* The first computer programs were written in machine code and assembly language. These arelow-level programming languages
* Nearly all web pages are made up of three key ingredients ― HTML, CSS and JavaScript. HTML is used to mark up the content. CSS is the presentation layer, and JavaScript adds the interactivity.
* Brendan Eich creador of JavaScript the lenguage for the navigator browser.
* Nearly all web pages are made up of three key ingredients HTML, CSS and JavaScript. HTML is used to mark up the content. CSS is the presentation layer, and JavaScript adds the interactivity.
* The old code must work the same way when interpreted by an engine running a new specification (it's a bit like saying that PlayStation 4 must still be able to run games created for PlayStation 1, 2 and 3).

Practice:

[ht}tps://codepen.io/michell17/pen/xxgNVRN](https://codepen.io/michell17/pen/xxgNVRN)

Chapter II - Programming Basic.

* The syntax used by JavaScript is known as a C-style syntax because of its similarities with theC programming language.
* Data types in JavaScript: string,symbol, number,boolean,undefined,null.
* Any value that isn’t one of the primitive data types listed above is anobject. These include arrays, functions and object literals
* An operator applies an operation to a value, which is known as theoperand. A unary operator only requires one operand; for example: typeof=operator and “hello”= operand.
* Using a Constructor Function

You can also create a string object using the following constructor function:

new String("hello")<< [String: 'hello']

This will create a new string that is the same as the string literal 'hello', although it will be classed as an object rather than a primitive data type. For this reason it is preferable to use the string literal notation, not to mention it requires less typing to use literals!

* A method is an action that a primitive data type or object can perform. To call a method, we use the dot operator ( . ) followed by the name of the method, followed by parentheses (this is a useful way to distinguish between a property and a method ― methods end with parentheses). For example, we can write a string in all capital letters using the toUpperCase() method:

name.toUpperCase(); << 'ALEX

* Template Literals

Template literalsare a special types of string that were introduced in ES6. Template literals use the backtick character, ` , to deliminate the string, as shown in the example below:

`Hello!`;

This has the advantage of being able to use both types of quote mark within the string:

`She said, "It's Me!"`

They also allow interpolation of JavaScript code. This means that a JavaScript expression can be inserted inside a string and the result will be displayed, as can be seen in the examples below:

const name = `Siri`;

`Hello ${ name }!`;

<< 'Hello Siri!'

const age = 39;

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

<< 'I will be 40 next year'

The JavaScript expression is placed inside the curly braces with a $ character in front of them. This is then evaluated and the result is returned in the string output.

* Symbols

Symbols were introduced as a new primitive value in ES6. They can be used to create unique values, which helps to avoid any naming collisions.

Symbols are the only primitives that don't have a literal form. The only way to create them is to use the Symbol() function:

const uniqueID = Symbol();

It is recommended to add a description of the symbol inside the parentheses:

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

Because symbols are primitive values, the typeof operator should return a type of 'symbol':

typeof uniqueID;

<< 'symbol'

The description acts as a string representation of the symbol and is used to log the symbol in the console, making it useful for debugging purposes:

console.log(uniqueID);

<< Symbol(this is a unique ID)

You can manually access the description using the String() function:

String(uniqueID)

<< 'Symbol(this is a unique ID)'

* Logical operator

! (Logical NOT)

Placing the ! operator in front of a value will convert it to a Boolean and return the opposite value. So truthy values will return false , and falsy values will return true . This is known asnegation:

!true; // negating true returns false

<< false

!0; // 0 is falsy, so negating it returns true

<< true

Note: you can use doble negation to find out if a value is true or not.

&& (Logical AND)

Imagine a nightclub that only allows people inside if they are wearing shoes AND over 18. This is an example of a logical AND condition: anybody going into the club must satisfybothconditions before they are allowed in.

The logical AND operator works on two or more values (the operands) and only evaluates to true ifallthe operands are truthy. The value that is returned is thelasttruthy value if they are all true, or thefirstfalsy value if at least one of them is false:

'shoes' && 18; // both values are truthy

<< 18

'shoes' && 0; // returns 0 because it is falsy

<< 0

|| (Logical OR)

Now imagine that the club relaxes its rules and allows people in if they wear shoes OR they're over 18. This means they only have to satisfy one of the rules to be allowed in. This is an example of a logical OR condition.

The logical OR operator also works on two or more operands, but evaluates to true ifanyof the operands are true, so it only evaluates to false if both operands are falsy. The value that is returned is thefirsttruthy value if any of them are true, or thelastfalsy value if all of them are false:

'shoes' || 0;

<< 'shoes'

NaN || undefined; // both NaN and undefined are falsy, so undefined will be returned

<< undefined

Practice:

<https://codepen.io/michell17/pen/RwKzjQL>

Chapter III – Array, logic and loop

* Removing values from an array:

Example:

Const example = [“hi”, “ok”,“bye”,];

Delete example[1];// that value Will be replace for undefined

* Multidimensional Arrays

You can even have an array of arrays, known as a multidimensional array. This could be used to create a coordinate system, for example:

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

* Sets

Sets were introduced to the specification in ES6. A set is a data structure that represents a collection of unique values, so it cannot include any duplicate values. Sets offer a useful way to keep track of data without having to check if any values have been duplicated. It's also quick and easy to check if a particular value is in a set, which can be a slow operation if an array is used.

Créating sets:

An empty set is created using the new operator and Set() constructor:

Const list = new set();

Adding values to sets:

Values can be placed into a set using the add method:

list.add(1); << Set { 1 }

Multiple items can be added to the set by repeating the add() method:

list.add(2).add(3).add(4); << Set { 1, 2, 3, 4 }

If any values are repeated in the array, then they will only appear once in the set:

const moreNumbers = new Set([7,7,7,7,7,8,8,8,9,9]);

moreNumbers << Set {7,8,9}

* Maps

Mapswere another data structure introduced in the ES6 specification. They are a convenient way of keeping a list of key and value pairs, and are similar to 'hashes', or 'hash tables' or 'dictionaries' in other programming languages.

How we can use Maps?

Was difficult to find about this but I found that we can use the method map to join array sor to multiply a length array with a number instead of use a loop.

* I understand the loops concept and how we can use that but was useful t oread how we can use a loop with map and with set is very different in how we use a loop in array

Example:

for(const letter of letters) {

console.log(letter);

}

<< h

e

l

o

it was a Good example in how we can use the loops in a set or map.

Practice:

<https://codepen.io/michell17/pen/VwPJyjx>

Chapter V – Function

* In JavaScript, functions are considered to be first-class objects. This means they behave in the same way as all the other primitive data types and objects in the language. They can be be assigned to variables, stored in arrays and can even be returned by another functions.
* Arrow Function

This ES6 syntax for declare a function is new for me barely I understand was Good to understand.

Arrow functions can be identified by the 'arrow' symbol, => that gives them their name. The parameters come before the arrow and the main body of the function comes after. Arrow functions are always anonymous, so if you want to refer to them, you must assign them to a variable. For example, the square function we wrote earlier can be written like so:

const square = x => x\*x;

Arrow functions have a number of advantages over other ways of declaring 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).

Example:

const tax = (salary) => {

const taxable = salary - 8000;

const lowerRate = 0.25 \* taxable;

taxable = taxable - 20000;

const higherRate = 0.4 \* taxable;

return lowerRate + higherRate;

}

Practice:

<https://codepen.io/michell17/pen/YzNoYQO>