What I have Learnt So far on My Javascript Track

I am Joy Sunday Egbala, currently in SCA3 stream, and I picked a carrier path in Javascript. So far, my experience has been awesome, I have learnt quite a lot of things and discovered new ways to get resources for JavaScript classes.

I have covered the following areas for the past 1 Month

* programming concept
* Basic JavaScript
* Basic Data structure
* ES6
* Version control

# **Programming Concepts**

Computer programming is the act of writing computer programs. A computer Program is a sequence of instructions written using a Computer Programming Language to perform a specified task by the computer. The act of writing computer programs is called computer programming.

Computer Programming Languages are also made of several elements. These basic elements include −

* Programming Environment
* Basic Syntax
* Data Types
* Variables
* Keywords
* Basic Operators
* Decision Making
* Loops
* Numbers
* Characters
* Arrays
* Strings
* Functions
* File I/O

To start learning programming especially JavaScript languages, you will need the following tools:

* A text editor to create computer programs. A text editor is a software that is used to write computer programs.
* A compiler: The conversion from plain text on your text editor to binary file is done by a Compiler and this process of conversion from plain text formatted program to binary format file is called program compilation. Finally, you can execute binary file to perform the programmed task.
* An interpreter: some programming languages such as Python, PHP, and Perl, do not need any compilation into binary format, rather require an interpreter to read their programs line by line and execute them directly without any further conversion.

## Programming Concepts Logic and Structure

### **Algorithm**

For Programming, an **algorithm** is a step-by-step procedure to resolve any problem.

### **Data Types and Variables**

Data type represents a type of the data which you can process using your computer program. It can be numeric, alphanumeric, decimal, etc.

Variables are the names you give to computer memory locations which are used to store certain values in a computer program. Example Int age. In this case, age is a variable of type integer which will store integer values in a certain location in the computer memory

While creating variables there are some key points you need to keep in mind −

* A variable name can hold a single type of value. For example, if variable **a** has been defined **int** type, then it can store only integer.
* Some programming language requires that variable are created before usage. You cannot use a variable name in your program without creating it, though programming language like Python allows you to use a variable name without creating it.
* You can use a variable name only once inside your program. For example, if a variable **a** has been defined to store an integer value, then you cannot define **a** again to store any other type of value.
* There are some words set aside by different programming language which perform different tasks when seen in a program, this words are called keyword. While naming variables, you cannot name a variable with the name of a keywords supported by that particular programming language. For example, if you will try to use any reserved keyword for the purpose of variable name, then you will get a syntax error. e.g. of some key words in C are main, printf, scanf, int, char etc.

### **Classes and Objects**

A class is a blueprint for creating objects. It consist of member functions and member variables. Also an object is an instance of class. One can access class member functions and variables with the help of object.

#### **Constructors**

This is a special method that is used to initialize the newly created object and is called just after the memory is allocated by the object.

class Person{

constructor(name){

console.log('Hello from' + ' ' + name);

}

}

const p = new Person('karan'); // Output Hello from karan

#### **Static Methods**

They belong to the class itself. They can be used without the instance (object) of a class. They don’t modify the behaviour of our class but is very powerful when want to view certain data etc Just add static in-front of a function to make it a static function.

#### **Recursion**

Recursion is a technique where a function calls itself. There is a termination condition called the base condition which stops the recursive action.

### **Loops**

Loops in programming languages is a feature which facilitates the execution of a set of instructions repeatedly while some condition evaluates to true.

let j = 0;

while(j < 10){

console.log(j]);

j++;

}

### **OOPS**

It is a methodology or paradigm to design a program using classes and objects. Main Components of OOPS are Encapsulation, Abstraction, Inheritance and Polymorphism. Lets look at them closely —

* **Encapsulation:** Encapsulating everything about an entity inside a class. i.e. Putting functions and data members inside a class.
* **Abstraction**: It simply means that you are hiding the important things inside the class and provides only the necessary or relevant information that you want to show through an object. Abstraction is simply data hiding.
* **Inheritance:** It is a concept where a class acquire properties of a child class. We can extend this child class and use already defined methods that are in the base class / Parent class. The main purpose of inheritance is to provide reusable code.
* **Polymorphism:** It generally means ability to take many forms. There are generally 2 types of Polymorphism — compile time and run time.
  + **Compile time*:*** Function Overloading is an example of compile time polymorphism. Same function name different parameter data types
  + **Run time*:***Function Overriding is an example of run time polymorphism.

### **Libraries/Modules/Packages**

It is just a function or list of functions that are wrapped around or are written for repetitive work. Every programmer should know how to create a library else he would spent a lot of time in writing same code again and again.

# **Basic JavaScript**

JavaScript is the programming language of the Web. JavaScript (JS) is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat.

To learn JavaScript, you must first be conversant with:

   1. **HTML**to define the content of web pages

   2. **CSS** to specify the layout of web pages

**JavaScript** is then added to these two to program for the behavior of web pages

JavaScript accepts both double and single quotes for string characters

## What can JavaScript be used for

* JavaScript Can be used to Change HTML Content

One of many JavaScript HTML methods is getElementById().

Example

document.getElementById("demo").innerHTML = "Hello JavaScript";

* JavaScript Can be used to Change HTML Attribute Values
* JavaScript Can Hide HTML Elements

Hiding HTML elements can be done by changing the display style:

Example

document.getElementById("demo").style.display = "none";

* JavaScript Can Show HTML Elements

Example

document.getElementById("demo").style.display = "block";

# Basic Data Structure

Data Structures are a specialized means of organizing and storing data in computers in such a way that we can perform operations on the stored data more efficiently.

## 1. Arrays

An array is a structure of fixed-size, which can hold items of the same data type. It can be an array of integers, an array of floating-point numbers, an array of strings or even an array of arrays (such as 2-dimensional arrays). Arrays are indexed, meaning that random access is possible.

Array operations

* Traverse: Go through the elements and print them.
* Search: Search for an element in the array. You can search the element by its value or its index
* Update: Update the value of an existing element at a given index

## 2. Linked Lists

A linked list is a sequential structure that consists of a sequence of items in linear order which are linked to each other. Linked List Data are accessed sequentially and random access is not possible.

Linked list operations

* Search: Find the first element with the key k in the given linked list by a simple linear search and returns a pointer to this element
* Insert: Insert a key to the linked list. An insertion can be done in 3 different ways; insert at the beginning of the list, insert at the end of the list and insert in the middle of the list.
* Delete: Removes an element x from a given linked list. You cannot delete a node by a single step. A deletion can be done in 3 different ways; delete from the beginning of the list, delete from the end of the list and delete from the middle of the list. It provide a simple and flexible representation of dynamic sets.

## 3. Stacks

A stack is a LIFO (Last in First Out — the element placed at last can be accessed at first) structure which can be commonly found in many programming languages.

* Push: Insert an element on to the top of the stack.
* Pop: Delete the topmost element and return it.
* Peek: Return the top element of the stack without deleting it.
* isEmpty: Check if the stack is empty.
* isFull: Check if the stack is full.

## 4. Queues

A queue is a FIFO (First In First Out — the element placed at first can be accessed at first) structure which can be commonly found in many programming languages.

* Enqueue: Insert an element to the end of the queue.
* Dequeue: Delete the element from the beginning of the queue.

## 5. Hash Tables

A Hash Table is a data structure that stores values which have keys associated with each of them. Furthermore, it supports lookup efficiently if we know the key associated with the value. Hence it is very efficient in inserting and searching, irrespective of the size of the data.

# **ES6**

ECMAScript 6 is also known as ES6 or ECMAScript 2015. It is the second major revision to JavaScript.

## New Features in ES6

**The let keyword**:

**let** keyword is used to declare a variable with block scope.

**Example**

var x = 10;

{

let x = 2;

// Here x is 2

}

// Here x is 10

**The const keyword:**

**const** keyword is used to declare a constant (i.e. a variable with a constant value).

Constants values cannot be changed.

**Example**

const x = 10;

### **JavaScript Arrow Functions:**

Arrow functions allows a short syntax for writing function expressions.

When writing an arrow function, you don't need the function keyword.

**Example**

const Multi = (x, y) => x \* y;

### **JavaScript Class:**

JavaScript Classes are templates for JavaScript Objects.

The **class** keyword is used to create a class.

When creating a class, always add a method named **constructor():**

**Example**

class Book {

constructor (author, year, title) {

this.author = author;

this.year = year;

this.title = title;

}

}

### **JavaScript Promise:**

A Promise is a JavaScript object that links "Producing Code" and "Consuming Code".

"Producing Code" can take some time and "Consuming Code" must wait for the result.

**Example**

let myPromise = new Promise(function(myResolve, myReject) {

setTimeout(function() { myResolve("I love You !!"); }, 3000);

});

myPromise.then(function(value) {

document.getElementById("demo").innerHTML = value;

});

### **JavaScript Symbol**

JavaScript Symbol is a primitive datatype just like Number, String, or Boolean.

It represents a unique "hidden" identifier that no other code can accidentally access.

**Example**

const book = {

Name: "John",

Author: "Doe",

age: 50,

eyeColor: "blue"

};

let id = Symbol('id');

person.id = 12321;

**Note:** Symbols are always unique. They can never be compared for equality, it will always return false

If you create two symbols with the same description they will have different values.

Symbol("id") == Symbol("id") // false

### **Default Parameter Values:**

ES6 allows function parameters to have default values.

**Example**

function myFunction(x, y = 10) {

// y is 10 if not passed or undefined

return x + y;

}

myFunction(5); // will return 15

### **Function Rest Parameter:**

The rest parameter (...) allows a function to treat an indefinite number of arguments as an array:

**Example**

function sum(...args) {

let sum = 0;

for (let arg of args) sum += arg;

return sum;

}

let x = sum(4, 9, 16, 25, 29, 100, 66, 77);

**Array.find():**

The find() method will return the value of the first array element that passes a test function.

This example finds will return the value of the first element that is larger than 25:

**Example**

var numbers = [90, 19, 21, 25, 30];

var first = numbers.find(myFunction);

function myFunction(value, index, array) {

return value > 25

}

Array.findIndex():

The findIndex() method will return the index of the first array element that passes a test function.

**Example**

var numbers = [4, 9, 16, 25, 29];

var first = numbers.findIndex(myFunction);

function myFunction(value, index, array) {

return value > 18;

}

### **New Number Properties:**

ES6 added the following properties to the Number object:

EPSILON: var x = Number.EPSILON;

MIN\_SAFE\_INTEGER: var x = Number.MIN\_SAFE\_INTEGER;

MAX\_SAFE\_INTEGER: var x = Number.MAX\_SAFE\_INTEGER;

### **New Number Methods:**

ES6 added 2 new methods to the Number object:

**Number.isInteger():** this method returns true if the argument is an integer.

**Example**

Number.isInteger(20); // returns true

Number.isInteger(7.1); // returns false

**Number.isSafeInteger():** A safe integer is an integer that can be exactly represented as a double precision number.The **Number.isSafeInteger()** method returns true if the argument is a safe integer.

**Example**

Number.isSafeInteger(200); // returns true

Number.isSafeInteger(12345678901234567890); // returns false

Safe integers are all integers from -(253 - 1) to +(253 - 1).

### **New Global Methods**

ES6 added 2 new global number methods:

**isFinite()** and **isNaN()**

**isFinite():** method returns false if the argument is Infinity or NaN else it returns true:

**Example**

isFinite(10/0); // returns false

isFinite(10/1); // returns true

**isNaN():** method will return true if the argument is NaN, else it returns false:

**Example**

isNaN("Hello"); // returns true

# **Version Control**

Using BitBucket and Git for Version Control: Working With Git has been a wonderful experience for me,

I Learnt how to create a repository on bitbucket, how to configure Git account on your local drive using git bash, clone your respository on your local drive, how to create files on your repository from git bash, how to navigate through directories and lots more