JAVASCRIPT

Developed by Alabian Solutions Ltd

CHAPTER 1

Introduction

Using HTML, a web page can be created and the web page can even be complex documents that intricately describe the content of a page. Adding CSS to the document produced by HTML, this can be presented in various ways like changing font size, typeface, background colour, image size, colour intensity, shadow etc. No matter how you dress it up, HTML and CSS can only achieve the static beauty of the web page. With JavaScript, you can bring that awesome web page to life. Effects such as slider, form validation, pop up, modal etc can be achieved with JavaScript.

Frontend development is the practice of producing HTML, CSS and JavaScript for a web page or web application so that a user can see and interact with them directly.

Brief history of JavaScript

1995: At Netscape, Brendan Eich created "LiveScript", which gets renamed to "JavaScript".
1996: Microsoft released "JScript", a port, for IE3.
1997: JavaScript was standardized in the "ECMAScript" spec.
2005: "AJAX" was coined, and the web 2.0 age began.
2006: jQuery 1.0 was released.
2010: node.JS was released.
2010: angular.JS was released.
2012: ECMAScript Harmony spec nearly finalized. What does the future hold for javaScript?
JavaScript and HTML5 technologies can be used to develop browser extensions, Windows 8 desktop widgets, and Firefox OS and Chrome OS applications. Many non-web related applications also use JavaScript as their scripting language. It can be used to add interactivity to PDF documents, create HTML templates (Mustache), interact with a database (MongoDB), and even control robots (Cylon.js)!

It certainly seems like JavaScript has a bright future. As the web platform continues to evolve and mature and its usage grows beyond the browser, JavaScript is sure to remain a central part of future developments.

Requirements to Learning JavaScript

- A basic knowledge of HTML and CSS
- A text editor
- A browser
- Logical thinking and tenacity

Programming

Computer programming is the process of developing and implementing various sets of instructions to enable a computer to do a certain task. It's simply about speaking the language of the computer.

Types of Programming Languages

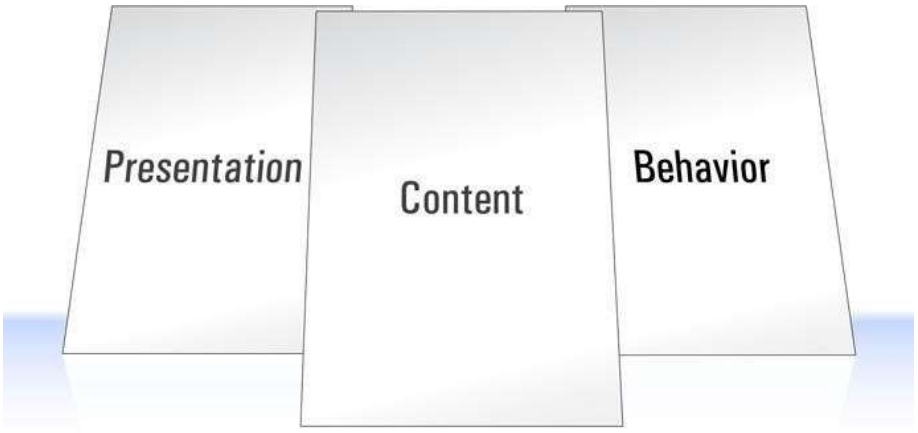
1. Low level language
    - Machine code
    - Assembly language
2. High level language

Computer programs are written in a language such as C, C++ or Java, PHP, JavaScript, C#, Python etc. These can either be compiled language or interpreted language. JavaScript is an example of interpreted.

Keeping Them Separate

HTML: It describes the content of the page.
CSS: It specifies the presentation of that content. JavaScript: It controls the behavior of that content.
While these technologies are used in producing a web page, they should be kept separate.

1. Create a separate .html file for HTML
2. Create a separate .css for CSS
3. Create a separate .js for JavaScript



The Wrong Approach

```
<p>

  Whatever you do, <a href="666.html" style="color: red;" onclick="alert('I said dont
  click')">don't click this link</a>!
```

The Right Approach

```
<p>

  Whatever you do, <a href="666.html" id="warning">don't click this link</a>! </p>


<style>
#warning{
color:red;

  }

</style>


<script>

let   warningLink=document.getElementById("warning");
warningLink.addEventListener('click',alerter, false);
```

A much better approach will be to save the style and script into a separate **.css** file and **.js** file and integrate them in the HTML document via link tag and script tag. That is the approach that will use in this manual.

```
<link rel="stylesheet" href="warning.css" /> <p>

  Whatever you do, <a href ="666.html" id="warning">don't click this link</a>!

 </p>

 <script src="warning.js"></script>
```

CHAPTER 2

JavaScript Basics

Statements

The syntax used by JavaScript is known as a C-style syntax, which is similar to the one used by PHP.

A JavaScript program is made up of a series of statements. These statements contain JavaScript language construct, keyword and command. In this manual we will learn these language constructs, keywords & command; and how we can use them to create awesome stuffs.

Each statement ends with a new line or semicolon.

But a popular convention adopted by JavaScript programmer is to write one statement per line and end the line with a semi colon.

```
document.write('Hello World') document.write('Here come the JS Engineer')

document.write('Hello World'); document.write('Here come the JS Engineer');


document.write('Hello World');
```

Comments

Comments are human-readable texts ignored by the computer. There are two types of comment in JavaScript:

```
//This is a single line comment


/*

 This is a multiple line comment on line one
 This is a multiple line comment on line two
```

Variable

JavaScript use "let" and "const" keywords to declare variables. The keyword **"const"** is used when the variable will not be reassigned to another value, whereas **"let"** is used if the variable might be reassigned later in the program.

Variable is a reference to a memory location in the computer memory (RAM) usually for storage of data. We perform three (3) actions with variable

1. Declare the variable
2. Put data inside the variable
3. Get the data inside the variable

To assign a value to a variable, we use the = operator. For example we declare a variable, called it name and another variable, and called it age:

Code:

```
const name = "Alabian";//this value cannot be changed
let age = 25;//this value can be change
```

Variables are usually given a name so that it is easy to differentiate them. And there are some rules that must be followed when giving variable name.

Variable naming rules

1. They can only contain the following character; letters, numbers, underscores or dollar signs but the first character cannot be a number
2. They are case sensitive so 'numberOne' is different from 'NumberOne'
3. They cannot be JavaScript reserved words.
4. Choose clarity and meaning
5. Since they cannot contain space variable name that compose of more than a word must be joined together. So 'number one' will be written as 'numberone'. But for ease of readability 'numberone' is usually written as 'numberOne' or 'number_one'. The camel case (ie numberOne) convention is preferred by many JavaScript programmers.
6. Pick a naming convention and stick with it.

```
let 2numberOne; //invalid variable name because it started with number

let numberOne!; //invalid variable name because it contains a non permitted character !

let x; //valid variable name but not descriptive

let numberOne; //variable declaration

let numberTwo; //variable declaration
numberOne=5; //assigning data to variable
numberTwo=10; //assigning data to
variable

document.write(numberOne); //calling data inside a variable
```

Scope

Scope is an important concept in programming. It refers to where a constant or variable is accessible by the program.
Using **const** and **let** to declare variables means they are **block scoped**, so their value only exists inside the block they are declared in.

What is a block scope?

A block scope is the area within **if**, **switch** conditions or **for** and **while** loops. Generally speaking, whenever you see {curly brackets}, it is a block. In ES6, **const** and **let** keywords allow developers to declare variables in the block scope, which means those variables, exist only within the corresponding block.

There are two types of scope in programs, which are; Global and Local scope.

Global Scope: Any variable declared outside of a block is said to have global scope. This means it is accessible everywhere in the program.

Local Scope: This means that any variables defined inside a block using the let or const will only be available inside that block and not be accessible outside of that block.

Data Types

JavaScript has five primitive data types, which are string, number, boolean, null and undefined. And one non-primitive data types which is object. There is also another data type, which is array but is a special form of object data type. Also string, number and boolean data types can also be expressed as object data type, more on this later in the manual.

String Data Type

A string is a collection of letters and symbols enclosed within quote mark. The quote mark can either be single or double.

If you want to use double quote marks inside the string, you need to use single quote marks to enclose the string. And if you want to use an apostrophe in your string, you need to employ double quote marks to enclose the string.

```
"Alabian Solutions"; //string

 'Alabian Solutions'; //same as above because double and single quote are same
"Alabian Solutions's ex-students are the best"; //single quote inside double quote"
'"Alabian Solutions is the best" - W3C'; //single quote inside double quote"



var academy="Alabian Solutions"; //assigning a string data into a variable
```

Another option to be used when placing single quote inside single quote or double quote inside double quote is to do what is called escaping the quotation mark. You place a backslash before the apostrophe so that it appears as an apostrophe inside the string instead of terminating the string:

```
'Alabian Solutions\'s ex-students are the best'; //single quote inside double quote"

"\"Alabian Solutions is the best\" - W3C"; //single quote inside double quote"
```

Number Data Type

Number can be integers (whole numbers, such as 3) or floating point decimals (often referred to as just "decimals" or "floats", such as 3.14159).

The only character permitted as number apart from digitals (0-9) are dot(.) and exponential(e or E). **NOTE** comma(,) is not a valid character in number in JavaScript.

```
289; //integer

45.6; //float

1e6; // means 1 multiplied by 10 to the power 6 (a million)

2E3; // can also be written as 2E3, 2E+3 and 2e+3
```

Boolean Data Type

There are only two boolean values which are **'true' and 'false'**. Boolean values are fundamental in the logical statements that make up a computer program. Every value in JavaScript has a Boolean value. While the list of those with falsy value is finite and is just seven, the list with truthy value is infinite. Below are the data with falsy value

"" // double quoted empty string '' // single quoted empty string

0

NaN false null

undefined

```
true;
false;

let isRich=true; //assigning boolean into a variable
```

Undefined and Null Data Type

Undefined is the value given to variables that have been declared but not yet assigned a value. It can also occur if an object's property doesn't exist or a function has a missing parameter.

While null represents a non-exiting data, more like a call to a variable that has not be declared before. So that variable does not represent any reference to any location in the computer memory.

While undefined is usually set by JavaScript engine, null is manually set by JavaScript programmer.

```
undefined;
null;

let justAVariable; //the value of this value before any assignment is undefined
```

Concatenation Operators (String) The only operation that can be performed on string data type is concatenation. The operation simply joins two strings data and produces another string which composes both strings.

```
"This is a " + " complete sentence"; //concatenating two literal

let partSentenceOne = "This is a ";

let partSentenceTwo = " complete sentence";
```

You can use Template Literals (which was introduced in ES6).

What is Template Literals?

Template literals or interpolation in other languages is a way to output variables in the string. We can use a new syntax ${variable_name} inside the back-ticked string:

```
let name = 'Mark';
let company_name = 'Alabian Solutions Ltd';

console.log(`${name} works at ${comapany}`);

Output: Mark works at Alabian Solutions Ltd
```

So with template literals no use of "+" sign operator to join one or two strings together.

Arithmetic Operators (Number)

All the usual arithmetic operations can be carried out in JavaScript.

```
let numberOne=4, numberTwo=2, numberThree;

numberThree = numberOne + numberTwo; // addition result is 6
numberThree = numberOne - numberTwo; // substraction result is 2
numberThree = numberOne/numberTwo; // division result is 2
numberThree = numberOne*numberTwo; // multiplication result is 8

numberThree = numberOne%numberTwo; // modulo division(reminder after division) result
is 0


++numberOne; // incrementer operator increases the value of numberOne to 5 from 4
```

Type Coercion

Type coercion is the process of converting the type of a value in the background to try and make an operation work. For example, if you try to multiply a string and a number together, JavaScript will attempt to coerce the string into a number.

```
"2" * 8; // the result is 16
```

This may seem useful, but the process is not always logical or consistent, causing a lot of confusion. For example, if you try to add a string and a number together, JavaScript will convert the number to a string and then concatenate the two strings together

```
"2" + 8; // the result is 28
```

Comparison Operators (All Data Type)

Use these operators to compare two values for equality, inequality, or difference.
The comparison operators are:

- Greater than >
- Less than <
- Greater than or equal to >=
- Less than or equal to <=
- Soft equality ==
- Strict equality ===
- Soft inequality !=
- Strict inequality !==

Code:

```
let quizScore = 10;

//using soft equality, this checks for only value
quizScore == 10; //true
quizScore == '10'; // true 10 == '10'; // true

//using strict equality, this checks value and type
quizScore === 10; // true
quizScore === '10'; // false

//using soft inequality, this checks only value
quizScore != 5; // true
quizScore != '5'; // true

//using strict inequality, checks for value and type
quizScore !== 5; // true
quizScore !== '5'; // false
```

Logical Operators (Boolean)

A logical operator can be used with any primitive value or object. The results are based on whether the values are considered to be truthy or falsy. These are typically used in combination with the comparison operators:

```
//Let's describe Mark ElliotZuckerberg

let isDeveloper = true;
let isAccountant = false;
let knowHTML = true;

let knowValuation = false;


//AND logical operator produces only true when both operands are true

isDeveloper && knowHTML; //true
isDeveloper && isAccountant; //false
knowValuation && isDeveloper; //false
isAccountant && knowValuation; //false


//OR logical operator produces true when either operand is true

isDeveloper || knowHTML; //true
isDeveloper || isAccountant; //true
knowValuation || isDeveloper; //true
isAccountant || knowValuation; //false
```

typeof Operator (All Data Type)

When typeof operates on its operand, the result it produces is the data type of the operand. It is a unary operator which implies it operates on one operand.

```
typeof 25; // number

typeof 'Alabian Solutions'; // string

typeof true; // boolean
```

CHAPTER 3

ARRAYS

Arrays

Why and what is array?

An array is a type of data-type that holds an ordered list of values, of any type. Counting of array values starts from zero (0).

Array literal

Using array literal is the one of the easiest way to create an array in Javascript. Each value are separated with a comma(,).

Syntax:

```
let array_name = [value1, value2, value3, …];
```

For example:

```
let color = [“blue”, “yellow”, “red”, “Green”];
```

Instead of we declaring each color value in a single variable like this:

```
let color1 = "blue";
let color2 = "Yellow";
let color3 = "Red";
let color4 = "Green";
```

Array helps us to solve that right.

N.B: Array accepts any data type (e.g. String, Number, Boolean, object etc.)

So how do we access the values in an array?

Accessing values in an Array

You can access an array value by referring to the **index number** inside the square bracket notation [].

And note, the index number starts form 0;

For example: to access the first value in the color array

```
Color[0];
```

Output:

```
=>Blue
```

Adding values to an Array

To add values to an array:

Lets create an empty array.

```
let cars = [];
```

Lets add values to the empty array

Code:

```
cars[0] = "Toyota";
cars[1] = "BMW";
cars[2] = "Honda";
cars[3] = "Lexus";
```

Lets check cars

```
Console.log(cars);
```

Output:

```
["Toyota", "BMW", "Honda", "Lexus"];
```

We can add additional values to the cars array by assigning it to an index number

```
Color[4] = "Benz"
```

Another way to add values to an array is to use the push() method and shift() method, which we will discuss later.

Changing array values

```
cars[0] = "Mazda"// this will change the first value "Toyota" to "Mazda"
```

Destructuring Arrays

Destructuring an array is the concept of taking values out of an array and presenting them as individual values. Destructuring allows us to assign multiple values at the same time, using arrays; for example;

Code:

```
const [x,y] = [4,5];
console.log(`x is ${x} and y is ${y}`);

Output:
X is 4 and y is 5
```

The Spread Operator (…)

The Spread Syntax

- The spread syntax is simply three dots: ...
- It allows the elements of an array to expand.

Code:

```
let mid = [3, 4];
let arr = [1, 2, ...mid, 5, 6];
console.log(arr);
//Output: [1, 2, 3, 4, 5, 6]
```

String to Array

You can use the spread syntax to convert a string into an array. Simply use the spread syntax within a pair of square brackets:

Code:

```
let str = "hello";
let chars = [...str];
console.log(chars);
//Output: ["h", "e", "l", "l", "o"]
```

Spread operator in function

Imagine you have a function, where you want to add three values:

```
function add(a,b,c) {
      return a + b + c;
}
```

Now imagine, you have an array of values:

```
const values = [1,2,3];
```

To put those values in the array inside the function, will do something like this

```
const sum = add(values[0], values[1], values[3]);
```

what happens when you have a function that will accept a dynamic number of arguments? Or even simpler yet, what if we have a scenario where this function wants 20 values and I give you an array with 20 numbers.

This is where u can use spread operator. Considering the function we are using. We will have:

```
const sum = add(…values);
console.log(sum);
```

Result:

```
=> 6
```

Array Properties and Methods

Every array has a length property. A length property determines how many items/values are in an array. And we can access the length property of an array by **dot** notation.

For example:

```
cars.length// this will return an integer or number that are in cars array
```

Output:

```
=>5
```

To access the last value in an array we can use the length property like this:

```
cars[cars.length-1] // accesses the last value in the cars array
```

Arrays methods: This perform an action on the array; either to change it or to tell us something about it. In addition, we can access the each method of an array by **dot** notation.

Code:

```
cars.pop(); // removes the last item in the cars array

cars.push('jeep'); // appends a new value to the end of the cars array

cars.shift(); // removes the first value from the cars array

cars.unshift('Hyndai'); // appends a new value to the beginning of the cars array
```

Multidimensional array

Arrays can be nested, meaning that an array can contain another array as an element.

```
let quizQuestions = [

      ['What is the capital of Lagos State?', 'Ikeja', 'Agege','Mushin','Ikeja'],
      ['What is the capital of Kaduna State?', 'Kagoro', 'Kachia', 'Kaduna',
      kaduna']
];
```

CHAPTER 4

LOGIC

if statement

Use if to specify a block of code to be executed, if a specified condition is true.

Syntax of the if statement

```
if (condition) {

//code to run if condition is true

}
```

Code:

```
const age = 12;
if (age < 18) {
      console.log('Sorry, you are not old enough to play');
}
```

if and else statement

Use else to specify a block of code to be executed, if the same condition is false.

Syntax of the if… else statement

```
if (condition) {
// code to run if condition is true
} else {
// code to run if condition is false
}

For example: Lets try and change the value of age to 27;
Const age = 27
if (age < 18) {
      console.log('Sorry, you are not old enough to play');
}else {
      console.log('Yes!!! You can play the game');
}
```

Ternary Operator

This is an alternative way or shorthand in writing the if … else statement.

Syntax: condition? Expression 1 : Expression 2;

Expression 1 will execute if the condition is true while

Expression 2 will execute if the condition is false.

For example

```
age < 18 ? console.log('Sorry, you are not old enough to play') :
console.log('Yes!!! you can play');
```

if…else if statement

Use else if to specify a new condition to test, if the first condition is false

Code:

```
let number = 5;
if (number === 4) {
console.log('You rolled a four');
} else if (number === 5) {
console.log('You rolled a five');
} else if(number === 6){
console.log('You rolled a six');
} else {
console.log('You rolled a number less than four');
}
```

Output

```
=>   'You rolled a five
```

CHAPTER 5

LOOP

Loops will repeat a piece of code repeatedly according to certain conditions.
N.B: A loop should have when to start, stop and how to get to the next item

while loop

Syntax for while loop

```
while (condition) {

//statement to repeat
//increment or decrement

}
```

For example: This example will countdown from 10 to 1

Code:

```
let count = 10;
while (count >= 1) {
      console.log (count);
      count = count -1;//or count --
}
```

for loop

for loop is used to repeatedly run a block of code - until a certain condition is met

Syntax for 'for loop'

```
for (initialize;  condition;  update) {
// statements to repeat
}

for (index = 0; index < 13; index++){
      console.log (index);
}
```

Nested for Loops

You can place a loop inside another loop to create a nested loop. It will have an inner loop that will run all the way through before the next step of the outer loop occurs.

For example: We can create a multiplication table

Code:

```
// 2-4 times table
      for (let x = 2; x <= 4; x++) {
            for (let y = 1; y <= 12; y++) {
                  let result = x * y;
                  document.write(`${x} x ${y} = ${result} <br>`);
            }
            document.write("<br>")
      }
```

Result:

While **x** represents the outer loop and **y** represents the inner loop. The outer loop runs according to the condition and enters the inner loop, so the inner loop runs until its condition its no longer true then it comes out and run the outer loop, this will continue till all conditions are meant.

iterating over an array

Use a for loop to easily process each item in an array:

For example:

```
let quizQuestions = [

['What is the capital of Lagos State?', 'Ikeja', 'Agege','Mushin','Ikeja'], ['What
is the capital of Kaduna State?', 'Kagoro', 'Kachia', 'Kaduna', 'Kaduna']
];
```

Code:

```
for (let i = 0; i < quizQuestions.length; i++) {
            var arrayAsElement = quizQuestions[i];
            for (let x = 0; x < arrayAsElement.length; x++) {
                  let innerElements = arrayAsElement[x];
                  console.log(innerElements);
      }
}
```

CHAPTER 6

FUNCTIONS

Function declaration and expression

To define a function literal we can use a function declaration:

```
function quizApp(){
      alert('My First Quiz App');
}
```

We can also use function expression to create a function literal

```
let quizApp = function () {
      alert('My First Quiz App');
}
```

Invoking a Function

Invoking a function is to run the code inside the function's body. To invoke a function, simply enter its name, followed by parentheses. Let's invoke the quizApp function **quizApp();**

Return values

All functions return a value, which can be specified using the return operator.

```
function quizApp() {
      return alert('My First Quiz App');
}

console.log(quizApp());
```

Parameters and Arguments

Parameters and arguments are often used interchangeably to represent values that are provided for the function to use. Let's break this down, parameters are placeholders holding down a position , while arguments are values supplied.

Declare two parameters, number1 and number2

Code:

```
function addNumbers(number1, number2){
      let result = number1 + number2;
      console.log(result);

}
```

Then, invoke the addNumbers() with arguments 5 and 10

```
addNumbers(5, 10);
```

Default Parameters

These are values that will be used by the function if no arguments are provided when it is invoked. To specify a default parameter, simply assign the default value to it in the function definition.

For example: A function calculating a discounted price in a store. This function takes two arguments: the price of an item and the percentage discount to be applied. The store's most common discount is 10%, so this is provided as a default value

Code:

```
function discount(price, amount=10) {
      return price*(100-amount)/100;
}
discount(500)// this will use the default value of amount
```

Arrow Functions

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, getting the square of a number

Code:

```
Const square = x => x*x;
```

N.B: if its only one parameter, parentheses is not needed but if it is more than one parameter, parentheses is needed and if there wont be any parameter, empty parentheses is needed. For example:

```
//more than one parameter
const add = (x,y) => x + y;// this adds two numbers together
//empty parentheses
const hello = () => alert('Hello World!');
```

Callbacks

A function that is passed as an argument to another is known as a callback. Simply, a callback function is a function that will be executed just after another function has finished executing.

```
let firstName = 'John';
let lastName = 'Omoluabi'
function greetings(name) {
      return `Good morning, ${name()}`;
}
function fullName(){
      return `${firstName} ${lastName}`;
}
console.log(greetings(fullName));
```

Classes

Classes are in fact "special functions", and just as you can define function expressions and function declarations, the class syntax has two components: class expressions and class declarations.

Class Declaration

To declare a class, you use the **class** keyword with the name of the class

Code

```
class person {}
```

One important thing to note here is that unlike function declarations, class declarations can't be hoisted. You first need to declare your class and then access it otherwise you get a **ReferenceError**.

Code:

```
let John = new person(); // ReferenceError
class person {}
```

Class expressions

A class expression is another way to define a class. Class expressions can be named or unnamed. The name given to a named class expression is local to the class's body.

Code:

```
let person = class {
       constructor() {
}
```

Constructor

The constructor method is a special method for creating and initializing an object created with a class. There can only be one special method with the name "constructor" in a class.

Code:

```
class Person {
       constructor (name, dob) {
            this.name = name;
            this.dob = dob;
   }
}
```

Creating an Instance of a Class

We create an instance of the class by using the **new** keyword.

Code:

```
class Person {
      constructor (name, dob) {
             this.name = name;
             this.dob = dob;
      }
      sayName() {
            console.log(`Hi! my name is ${this.name} and my birthday is on
            ${this.dob}`);
      }
}

let mark = new Person("mark", "July 10");
console.log(mark.sayName())
```

Result:

```
Hi! my name is mark and my birthday is on July 10
```

Methods in Classes

There are two types of methods that can be created using class: The prototype methods and static methods.

Prototype Methods

The method **sayName** used in our example above is a prototype method. Prototype methods can be called by an instance of a class. Prototype methods also include getters and setters.

Static Methods

Static methods cannot be called by instances of a class. They are only available to a class that is called without creating an instance of it. If you call a **static method** from an instance, you will get an error. This is created using the static keyword.

Code:

```
class Person {
	constructor (name, dob) {
		this.name = name;
		this.dob = dob;
	}
	static sayName(data) {
		console.log(`Hi! my name is ${data.name} and my birthday is on
		${data.dob}`);
	}
}
let Stanley = new Person("mark", "July 10");
console.log(Stanley.sayName())// This will throw an error

Call a static function like this:
let John = new Person('John', '12th of April');
Person.sayName(John);
```

Inheritance

Use the "extends" keyword to create a class inheritance and the new class inherits all the methods in the parent class. To access the parent class' methods and properties, call the super() method in the constructor() method.

```
//Human class (parent class)
class Human{
	constructor(legs){
		this.legs = legs
	}
	humanLegs(){
		return `Human has ${this.legs} legs`;
	}
}

//Person class (inheriting Human class' methods and properties)
class Person extends Human{
	constructor(legs, gender){
		super(legs)
		this.legs = legs;
		this.gender = gender
	}
	description(){
		return `${this.humanLegs()}. I am a ${this.gender}`
	}
}
let Doe = new Person(4, 'male');
console.log(Doe.description());
```

CHAPTER 7.

OBJECTS

What is object?

Objects are a data type that let us store a collection of properties and methods.
An object in JavaScript is quite similar to an **array** — it is a data type, which stores lots of values. These values can be any data type — numbers, strings, booleans, functions, and even arrays and objects!
The difference between an array and an object is how the values are referenced. In an array, you reference a value with a number (its position in the array i.e its index) while we reference a value from an object by its ***name*** instead of a number

Object literals

An object literal is an object that is created directly in the language by wrapping all its properties and methods in curly braces {} and stored in a variable.

Creating object

Code:

```
const person = {
      age: 25,
      name: "John",
      weight: 25,
      hobbies:['Travelling', 'Dancing', 'Reading'],
      car:{name:'Honda Accord', colour:'black'},
      walk: ()=> alert(`${person.name} is walking`)
};
```

We call this a ***key-value pair.*** So an object is made of key-value pairs separated by commas, and the whole thing is wrapped in a set of curly brackets

Accessing properties

You can access object properties using dot notation or bracket notation

Using dot notation

```
person.age; person.name;
```

Using bracket notation

```
Person["age"]; person["name"];
```

Calling methods

```
person.walk();
```

Changing objects

All objects are mutable i.e the properties and methods can be change or removed and new properties and methods can be added.

Code:

```
person.age = 32;// this will change the value of age in the object
console.log(person.age)
```

output:

```
=> 32
```

Adding and deleting properties

Adding properties

Using the example above, lets add a height property to the object person

Code:

```
Person.height = "6 inch";
```

Deleting properties

We can use the keyword delete to remove a property

```
delete person.weight;
```

Array of objects

Since arrays can hold any data type, they can also hold objects:

```
const quizApp = [

      {question: 'What is the capital of Haiti?', optionA: 'Venatu', optionB:
      'Port-au-Prince', answer: 'Port-au-Prince'},

      {question: 'What is the capital of Nepal?', optionA: 'Havana ', optionB:
      'Juba ', answer: 'Havana'}

];
```

Access array of object like thus:

```
quizApp[0].question;
quizApp[1].question;
```

Object Destructuring

Destructing helps to write less code and easy way to access the properties in JavaScript.

For example:

Code

```
const person = {
      name: 'John Doe',
      country: 'Canada'
};
const { name, country} = person;
console.log(`I am ${name} from ${country} and I am ${age} years old.`);
```

Nested Object Destructuring

The following code snippet shows how we can use nested object destructuring:

```
const student = {
      name: 'John Doe',
      age: 16,
      scores: {
            maths: 74,
            english: 63
      }
};
```

We define 3 local variables: name, maths, science

```
const { name, scores: {maths, science = 50} } = student;
console.log(`${name} scored ${maths} in Maths and ${science} in Elementary
Science.`);
```

Math objects

The Math object has several properties that represent mathematical constants and methods.

All the properties and methods of the Math object are immutable and unable to be changed.

Mathematical Constants (properties)

Contants	Meaning
Math.PI	The ratio of the circumference and diameter of a circle
Math.SQRT2	The square root of 2
Math.SQRT1_2	The reciprocal of the square root of 2
Math.E	Euler's constant
Math.LN2	The natural logarithm of 2
Math.LN10	The natural logarithm of 10

Math.LOG2E	Log base 2 of Euler's constant
Math.LOG10E	Log base 10 of Euler's constant

Mathematical operation (methods)

Operators	Meanings
Math.floor()	The Math.floor() method will round a number down to the next integer, or remain the same if it is already an integer
Math.round()	The Math.round() method will round a number to the nearest integer
Math.pow()	The Math.pow() method will raise any number (the first argument) to the power of another number (the second argument)
Math.sqrt()	The Math.sqrt() method returns the positive square root of a number:
Math.max()	The Math.max() method returns the maximum number from its arguments
Math.min()	And the Math.min() method unsurprisingly returns the minimum number from the given arguments
Math.log()	The Math.log() method returns the natural logarithm of a number

Date object

Date objects hold information about dates and times. A constructor function is used to create a new date object using the new operator. Get today's time

Code:

```
Const today = new Date();
```

If an argument is not supplied, the date will default to the current date and time. The parmeters that can be provided are as follows:

New Date(year, month, day, hour, minutes, seconds, milliseconds)

Getting Methods

- getDate(): This is used to find the day of the week. It returns a number starting from zero(0) i.e Sunday will return 0, Monday will return 1 etc.
- getDate(): This returns the day of the month.
- getMonth(): tis returns an integer starting from zero(0) i.e January is 0, Febuary is 1 etc
- getFullYear(): This returns the year in 4 digits e.g 2018.
- We have the following methods; getHour(), getMinutes(), getSeconds(), getMilliseconds().

Read More

To understand more on object. Click on the links below

https://dev.to/codetheweb/javascript-objects-2c18

https://www.w3schools.com/js/js_objects.asp

CHAPTER 8.

DOCUMENT OBJECT MODEL (DOM)

Introduction to DOM

The HTML DOM is a standard **object** model and **programming interface** for HTML. The Document Object Model, or DOM for short, represents an HTML document as a network of connected nodes that form a tree-like structure.

DOM access

On every webpage, the document object gives us ways of accessing and changing the DOM. Every DOM "node" has properties that let us traverse the DOM like a tree: **parentNode, childNodes, firstChild ...**

```
Let bodyNode = document.body;

Let htmlNode = document.body.parentNode;

for(let i=0; i<document.body.childNodes.length;i++){
      var childNode = document.body.childNodes[i];
}
```

Getting an element by its id

This method **getElementById()** returns a reference to the element with a unique Id attribute that is given as an argument.

Code:

```
<div id='quizContainer'>Quiz App</div>

quizContianer = document.getElementById('quizContainer');
```

Getting elements by their tag names

This is will return all the live node list of all the elements with the tag name that is provided as an argument

Code:

```
<div id='quizContainer'>
      <h1 class='quizTitle'>Quiz App</h1>
      <li><input type="radio" name="quizRadio" class="inputClass"
      value="A">Question one</li>
      <li><input type="radio" name="quizRadio" class="inputClass"
      value="B">Question two </li>
      <li><input type="radio" name="quizRadio" class="inputClass"
      value="C">Question three</li>
```

```
      <li><input type="radio" name="quizRadio" class="inputClass" value="D">
      Question four </li>
</div>

let inputTag = document.getElementsByTagName('input');
```

The inputTag variable behaves like an array, as we can see below.

```
inputTag[0].value; //getting the value in the first input tag
inputTag[2].value; // getting the value in the third input tag
```

We can also have or get access to the values in the input as we've seen above using the value property

Getting elements by their class names

This is the use of getElementsByClassName() method which will return every elements that has a class name that is supplied as an argument.

```
inputClass = document.getElementsByClassName('inputClass');
inputClass[0].value;
inputClass[3].value;
```

Query selector

The document.querySelector() method allows you to use CSS notation to find the first element in the document that matches a CSS query selector criteria provided as an argument.

```
let quizContainer = querySelector('#quizcontainer');
```

The document.querySelectorAll() method also uses CSS notation but returns a node list of all the elements in the document that match the CSS query selector.

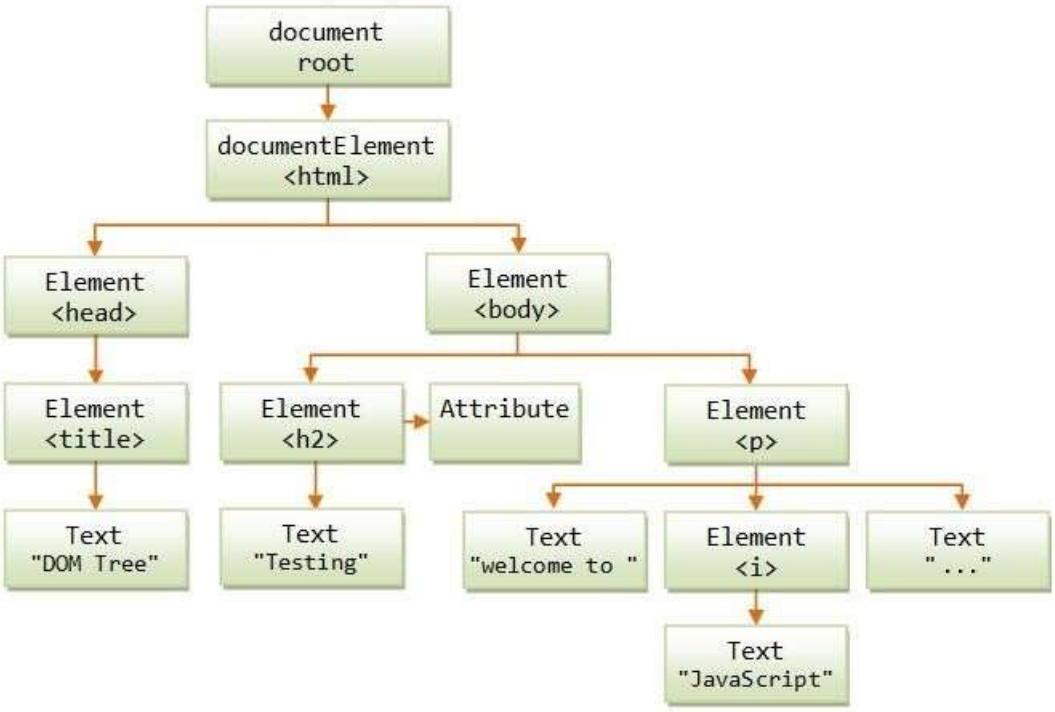
```
var inputSelector = document.querySelectorAll('.inputClass');
```

CSS query selectors are a powerful way of specifying very precise items on a page.

```
let heading = document.querySelectorAll('#quizContainer h');
```

Navigating the DOM

DOM TREE DIAGRAM



Getting and setting element's attribute

The setAttribute can change the value of an element's attributes. It takes two arguments: the attribute that you wish to change and the new value of that attribute.

```
h1.setAttribute('id', 'quizTitle');
```

The getAttribute() method returns the value of the attribute provided as an argument.

```
quizTitle = h1.getAttribute('id');
```

Updating the DOM by creating dynamic markup

The document object has a createElement() method that takes a tag name as a parameter and returns that element.

```
let newPara = document.createElement('p');
```

A text node can be created using the document.createTextNode() method.

```
let text = document.createTextNode('You have five questions to answer');
```

Next, append the text to the newPara.

```
newPara.appendChild(text);
```

Also, append newPara to the quizContainer

```
quizContainer.appendChild(newPara);
```

You can also insert element before a specified element using the insertBefore() method

```
<ul id=' items' >
      <li>Item one</li>
      <li id='itemTwo'>Item two</li>
      <li>Item three</li>
</ul>
let itemTwo= document.getElementById('itemTwo');
let items = document.getElementById('items');
items.insertBefore(newPara, itemTwo);
```

Changing the CSS of an element

Every element node has a style property. These can be used to dynamically modify the presentation of any element on a web page.

Any CSS property names that are separated by dashes must be written in camelCase notation, so the dash is removed and the next letter is capitalized because dashes are not legal characters in property names.

```
quizContainer.style.backgroundColor = 'blue';
quizTitle.style.textAlign = 'center';
```

Browser object model Window Object

The Browser Object Model (or BOM for short) is a collection of properties and methods that contain information about the browser and computer screen. It can be used to determine:

- The users browser.
- Dimensions of their screens.
- Pages visited before the current page.
- Creating pop-ups windows.

For example:

userAgent property will return information about the browser and operating system being used.

```
window.navigator.userAgent
```

The window.screen object contains information about the screen that the browser is displayed on.

```
window.screen.height;
```

```
window.screen.width;
window.screen.availWidth;
window.screen.availHeight;
```

window location

The window.location property is an object that contains information about the URL of the current page.

```
window.location.href;
window.location.hostname;
window.location.pathname;
window.location.protocol;
window.location.assign()
```

Window Popups

1. The window.alert() method will pause the execution of the program and display a message in a dialog box. The message is provided as an argument to the method, and undefined is always returned. For example window.alert("hello world").
2. The window.confirm() method will stop the execution of the program and display a confirmation dialog that shows the message provided as an argument, and giving the options of OK or Cancel. It returns the boolean values of true if the user clicks OK, and false if the user clicks Cancel: for example window.confirm('Do you wish to continue?');
3. The window.prompt() method will stop the execution of the program. It display sa dialog that shows a message provided as an argument, as well as an input field that allows the user to enter text. This text is then returned as a string when the user clicks OK. If the user clicks Cancel, null is returned:

Timing functions

setTimeOut(): The window.setTimeout() method accepts a callback to a function as its first parameter and a number of milliseconds as its second parameter.

```
function alertBox (){

      alert('It\'s quiz time');

      let timer = setTimeout(timing, 3000);

}
```

The window.setInterval() method works in a similar way to window.setTimeout(), except that it will continue to invoke the callback function after every given number of milliseconds.

Code:

```
function alertBox (){

      alert('It\'s quiz time');

}

let callAlert = setInterval(alertBox, 1000);
```

To stop at an animation at a certain point, store the timer into a variable and clear with one of these methods:

- window.clearTimeout(timer);
- window.clearInterval(callAlert);

CHAPTER 9.

EVENT LISTENER

Event listeners are like notifications to alert you when something happens. Every time an event occurs, a callback function will be called. Event listeners are added to elements on the page that are are part of DOM API.

Syntax

```
element.addEventListener(event, function, useCapture);
```

Parameters in addEventListener()method

Event: This is the first parameter that shows the type of event e.g. click, mouseover, etc

Function: This is a callback function that is called when an event occurs.

UseCapture: This is a Boolean value that specifies whether capturing should be used or not. It defaults to false, which is why bubbling happens by default. Capturing and Bubbling are two types of event propagation. An event is said to propagate as it moves from one element to another. Bubbling occurs when an event is fired on an element and it bubbles up the document tree firing the event on the parent element until reaches the root node. This is the default behavior of an event, which has a Boolean value of ***false***. Capturing occurs by firing an event on the root element, then propagates downwards, firing the event on each child element until reaches the target element that was clicked on.

Adding event listener

The addEventListener() method is called on a node object, the node to which the event listener is being applied. For example, this code will attach an event listener to the document's body.

```
<div id="count"></div>

<button id="counter">Counter</button>

<script>

      let button = document.getElementById("btn");
      let count = document.getElementById("count");
      let pos = 0
```

```
      button.addEventListener("click", clickMe,false);
      function clickMe() {
            count.innerHTML = ++pos;
      }
</script>
```

Event Types

The browser triggers many events. A short list of mouse events

MouseEvent: mousedown, mouseup , click, dblclick, mousemove, mouseover, mousewheel, mouseout

TouchEvent: touchstart, touchmove, touchend, touchcancel keyboard events

KeyboardEvent: keydown, keypress, keyup form events: focus, blur, change, submit window events: scroll, resize, hashchange, load, unload

Event Properties

When your event listener is called, the browser passes an event object with information about the event into the function.

Each event type has different properties. For example, MouseEvent reports the clicked coordinates:

```
<img id="img" src=" https://alabiansolutions.com/images/alabian-bg.jpg">
<div id="info"></div>

<script>

      let img =  document.getElementById('img');
      let info    = document.getElementById('info');
      function onClick(event) {
            info.innerHTML = 'You clicked on mouse X coordinate' + event.clientX +
            ' , and mouse Y coordinate' + event.clientY;
      }

      img.addEventListener('click', onClick, false);

</script>
```

Removing event listeners

```
cartoonImg.removeEventListener('click', onClick, false);
```

Stopping Default Behaviour

Some elements have default behaviors associated with certain events. For example, when a users clicks on a link, the browser redirect to the site in the href attribute.

To prevent this, preventDefault() method is used inside the callback function to stop the default behavior from occurring.

Code:

```
<p>
      <a id="stopped" href="http://alabiansolutions.com">Stopped Link</a>
</p>

let stopped = document.getElementById("stopped");
stopped.addEventListener("click", function(event) {
      event.preventDefault(); console.log("Stopped Link!");
} );
```

Chapter 10.

Forms

Accessing form elements

The DOM has a method called document.forms that returns all the HTML collection of all forms in order they appear. Since it's a collection you have to use the index notation to return a specific form.

For example:

```
<form id="firstForm" name="search" action = "/controller.php">

      <label for="fname">First Name</label>

      <input type="text" id="fname" name="fname">

</form>

const form = document.forms[0]//this will return the first form in the
document
Or
const from = document.getElementbyTagName("form")[0];
Or
const form = document.forms.search// using the value of name attribute  of
the form
Or
Const form = document.forms["search"]// using the square notation which is
preferable.
```

You can access form element using the following:

A form object also has a method called **elements** that returns an HTML collection of all the elements contained in the form.

```
document.getElementById('firstForm').elements[0];

document.forms['firstForm'].elements[0];

document.forms.firstForm.elements[0];
```

Retrieving and changing values from a form

Text input element objects have a property called value i.e **input.value** that is used to find the text inside the field. It is also used to set the value of an input. E.g.

Input.value = "search here";

But we can use the HTML placeholder for this anyway.

Form properties and methods

form.submit() method will submit the form automatically
form.reset() method will reset all the values in the form controls
form.action property can be used to set the action attribute of a from, so that it is sent to a different URL to be processed on the server

Form Events

- **Focus event:** this occurs when the cursor is placed inside the input field by either clicking or navigating to it using the keyboard.
  For example:

```
<body>
	<form name="formEvent" action="">
		<label>Search: </label>
		<input type="text" name="fname">
		<button type="submit">Search</button>
	</form>
	<div id="container"></div>

	<script type="text/javascript">
		const container = document.getElementById("container");
		const form = document.forms["formEvent"];
		let input = form["fname"];
		let btn = form.elements[1];


		input.addEventListener("focus",(e)=>{
			e.preventDefault();
			container.innerHTML = input.value;
			console.log(`you focused on me`)
		} ,false)
	</script>
</body>
```

- **Blur event:** this occurs when a user moves away from the input
- **Change event:** this occurs when a user moves the focus from the form after changing it. If a users clicks and makes no changes, the change event wont fire up rather the blur event will.
- **Input event:** This event is similar to the change event. The difference is that the input event occurs immediately after the value of an element has changed, while change event occurs when the element loses focus, after the content has been changed. The other difference is that the change event also works on <select> elements.

Form validation

Form validation is the process of checking whether a user has entered the information into a form correctly. Examples of the types of validation that occur include ensuring that:

- A required field is completed
- An email address is valid
- A number is entered when numerical data is required
- A password is at least a minimum number of characters

1.

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>Form Validation: validate text field</title>
	</head>
	<body>
		<form name="myform" method="post" action="index.html"
		onsubmit="return validateTextField()" >

			<div>
				<label for="name">Name:</label>
				<input type="text" id="name" name="name">
			</div>
			<input type="submit" value="Submit">

		</form>
	<script>
		function validateTextField(){

			let name=document.myform.name.value;

			if(name.length<=1){
				alert("Name too short."); return false;
			}

			if (name==null || name==""){ alert("Please, fill the name
			field");
				return false;
			}

		}

	</script>

	</body>

</html>
```

2.

```
<!DOCTYPE html>

<html>

	<head>
		<meta charset="utf-8">

		<title>Form Validation: validate number field</title>

	</head>

	<body>

		<form name="myform2" onsubmit="return validateNumber()" > Number:
			<input type="text" name="number">
			<input type="submit" value="submit">

		</form>

		<script>
			function validateNumber(){
				let num = document.myform2.number.value;
				if (isNaN(num)){  alert("Only numeric values are
			allowed");
					return false;

				}else{

					return true;
				}
			}

		</script>
	</body>

</html>
```

JS Best Practices

Adopt a set of good, consistent coding conventions. Use a code compression tool like UglifyJS.

Use libraries when appropriate, like jQuery or Zepto.

Further Study

Online Tutorials (Codecademy)

Books (Eloquent JavaScript, Maintanable JavaScript)

Meetups

Conferences (jQuery Con, JSConf, Fluent, ...)