**JAVASCRIPT MANUAL**

**WHAT IS JAVASCRIPT ?**

* High level (memory management), interpreted programing language
* Conforms to the **ECMAScript** specification
* Multi-paradigm
* Runs on the client/browser as well as on the server (Node.Js)

**WHY IS JAVASCRIPT ?**

* It is the programming language of the browser
* Build very interactive user interfaces with frameworks like React, Angular and Vue
* Used in building very fast server side full stack application.
* Used in mobile development (React Native, NativeScript, Ionic).
* Used in desktop application development (Electron Js)

**WHAT YOU NEED TO LEARN IN THIS COURSE**

* Variables & Data Types
* Arrays
* Object Literals
* Methods for strings, arrays, objects
* Loops – (for, while, for..of, forEach, map)
* Conditionals (if, ternary & switch)
* Functions (normal & arrow)
* OOP (prototypes & classes)
* DOM Selection
* DOM Manipulation
* Events
* Basic Form Validation

**NB: For further learning visit some course online by netninja, w3schools etc.**

* **Ansync JS**
* **Fetch Api & Ajax**
* **Vanilla Js**
* **Frameworks courses.**

**Variables & Data Types**

Create a sandbox folder where you want to execute all your files.

You will need a html file to work with your JavaScript.

In your html file you can then link your JavaScript file using the <script> tag at top of the </body> closing tag.

They are two ways you can use javascript in your html file.

***Method one:***

You can add the file by just adding an opening tag and a closing tag of the script.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

    </script>

</body>

</html>

***Method two:***

This is the recommended way of adding your javascript with your html.

This method requires you to create a separate file and then link it together with the html file.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script src="/style.js"></script>

</body>

</html>

We can use the alert and console to print and for debug

alert('Hello world');

console.log('Hello There');

console.error('This is an error');

console.warn('This is a warning');

**Check mdn.documentation for console.**

**variables**

Let talk about the variables; we have the **var, let and const**.

**Var;** has been use since the beginning of javascript and you don’t want to use it because it is globally scoped during conditional statement (conflict issue).

**Let & Conts;** they were both added in ES6 and they are block level scope. Which was a new update in javascript that gave us a lot of functionality

They difference between let and const is that with **let** you can reassign values and with **const** you can’t

let age = 40;

age = 45; // you have reassign age

console.log(age);

const age = 40;

age = 45; // this will flag an erro

console.log(age)

always use **let** when you know you are going to reassign a variable and use **const** when you are not going to reassign a variable

**Data Types**

We have **primitive data types (**data is directly assign to memory)

Strings, number, Boolean, null, undefined, and symbol(this is not that common)

const name = 'Nathaniel';

const age = 20;

const rating = 4.5

const isCool = True;

const x = null;

const y = undefined;

console.log(typeof name)

**concatenation**

they are several ways of concatenating a string the old way is using the (+) sign

console.log('My name is ' + name + 'and i am ' + age +'years old');

***using the template string ( ` )***

This is the modern way of concatenation and it the recommended method.

console.log(`My name is ${name} and i am ${age} years old`);

**String properties and methods**

In using JavaScript properties and methods, it only have a property called the length and several methods. To know the difference between them, methods ends with ().

//  string properties and method

 const s = 'bag, shoe, belt,phone';

  //length, toUpperCase

 console.log(s.substring(0,5).toUpperCase())

 console.log(s.startsWith)

const str1 = 'Saturday night plans';

 console.log(str1.startsWith('Sat'));

//   expected output: true

console.log(str1.startsWith('Sat',3));

//  expected output: false

We have **reference data types**

**Arrays**

These are variables that hold multiple values. We have couples ways in creating arrays.

*Array constructor*

This is an old way of using array.

const numbers = new Array(1,2,3,4,5);

console.log(numbers)

This is the new way of using array.

const food = ['amala', 'eba' ,'rice','fufu']

in javascript we can have multiple datatype within an array

const food = ['amala', 'eba' ,'rice','fufu',30,true]

Arrays are zero base in every language, we can access the data and also reassign values to the data.

const food = ['amala', 'eba' ,'rice','fufu',30,true]

console.log(food[1])

const food = ['amala', 'eba' ,'rice','fufu',30,true]

food[2] = 'beans'

console.log(food)

we cannot reassign the whole array but you can manipulate it and use method on it using **const.**

const food = ['amala', 'eba' ,'rice','fufu',30,true]

food = [] // this will flag error

**Array properties and methods**

const food = ['amala', 'eba' ,'rice','fufu',30,true]

food.push('garri') //push a value to the end

food.unshift('mango') // this is adding value to the begining

food.pop() // this pops take away the last value off

food.reverse() // this reverse the data from right to left

food.splice(0,4) // this take of data from the array

// to check if something is an array

console.log(Array.isArray(food));

// getting the index of a data

console.log(food.indexOf('eba'));

console.log(food)

**Object Literals**

This is basically key and value pair

const person = {

    firstName: 'Nathaniel',

    lastName: 'Nosa',

    age: 30,

    hobbies: ['party','code','read','vacation','swim','music'],

    address: {

        street: 'Lagos,',

        state:'Sate'

    }

};

console.log(person)

// single value

console.log(person.firstName, person.lastName)

console.log(person.address.state);

// de-structuring

const {firstName, lastName, address:{street}} = person;

// console.log(street)

*Array of an objects*

// array of an oject

const todos = [

: true

    },

    {

        id: 2,

        task : 'Study bootstrap',

        isComplete: true

    },

    {

        id: 3,

        task : 'Study git and git hub',

        isComplete: true

    },

    {

        id: 4,

        task : 'Study javascript',

        isComplete: false

    },

]

console.log(todos[1].task);

***Little on Json***

Json is a full stack development using API,when sending data to a server we send in json format and receive in json format just as object literal ;

Search for *json converter: freeformaerter.com/jsonformater*

// array of an oject

const todos = [

    {

        id: 1,

        task : 'Study html and css',

        isComplete: true

    },

    {

        id: 2,

        task : 'Study bootstrap',

        isComplete: true

    },

    {

        id: 3,

        task : 'Study git and git hub',

        isComplete: true

    },

    {

        id: 4,

        task : 'Study javascript',

        isComplete: false

    },

]

const todosJson = JSON.stringify(todos)

console.log(todosJson);

**Loops**

This is use to iterate information either an array, object etc. we have several kind of loops that we will be considering in this course

*For loop*

for(let i=0; i<10; i++){

    console.log(`this is ${i}`)

}

*While loop*

let i =0;

while(i<10){

    console.log(`this is ${i}`)

    i++;

}

Let consider the todo list above and use the for loop

for(let i=0; i< todos.length; i++){

    console.log(todos[i]) //to get all the todos

    console.log(todos[i].task) // to get the task only

    console.log(todos[i].id) // to get the id

    console.log(todos[i].isComplete) // to get the completed tasks

}

They are other methods we can use to loop through this todos

*For…of loop*

for(let todo of todos){

    console.log(todo);

    console.log(todo.text);

    console.log(todo.id);

}

We also have the **high order array method** which is the recommended way and they take in a function and the call back function can take in multiple parameter.

*For ..each loop*

This just loops through them

todos.forEach(function(todo){

    console.log(todo);

    console.log(todo.task);

    console.log(todo.id);

})

*map*

This allows us to create a new array from an array, but similar to for..each

const tasks = todos.map(function(todo){

    return todo.task

})

console.log(tasks);

*filter*

This allows us to create an array base on condition.

const isCompletes = todos.filter(function(todo){

    return todo.isComplete===true

})

console.log(isCompletes);

you can also use the loops together

const isCompletes = todos.filter(function(todo){

    return todo.isComplete===true

}).map(function(todo){

    return todo.task;

});

console.log(isCompletes);

**Conditional**

As the word implies this is executing blocks of code base on certain conditions.

*If*

const x = 10;

if(x == 10){

    console.log("x is 10");

}

As far as the equal sign goes, if we use (= =) equal this will not match the data types only the value, but if we use the (= = =) this will match both the data type and the value.

*Else*

const x = 20;

if(x == 10){

    console.log("x is 10");

}else{

    console.log("x is not 10");

}

*Else if*

This is use for extra condition, ad we can have/use as many else..if as we want.

const x = 20;

if(x == 10){

    console.log("x is 10");

}else if(x===20){

    console.log("x is 20");

}else{

    console.log("x is not 10");

}

*multiple codition*

const x = 20;

const y = 30;

if (x > 5 && y < 25 ){

    console.log('x and y are both great values')

}

*Teneray condition*

This is just like a shorthand way of the if statement and use to assign variables base on condition.

const x = 10;

const color = x > 10 ? 'red' : 'blue';

console.log(color)

*switches*

This is another way to evaluate a condition.

const color = 'red'

switch(color){

    case 'red':

        console.log('the color is red');

        break;

    case 'blue':

        console.log('the color is blue');

        break;

    default:

        console.log('The color is not red or blue')

}

**Function**

Function is a block of code which only runs when it is called upon.

function addNum(num1, num2){

    console.log( num1 + num2);

}

addNum(5,6)

if we don’t pass in a value to the called function we will get NaN (not a number) in return, and with function we can set default value for the parameter.

function addNum(num1=3, num2=3){

    console.log( num1 + num2);

}

addNum()

but if we have a default value for the parameter and also pass in a value for the called function, the called function value with override the default value.

We don’t print in a function we return values in a function and print the function we called.

function addNum(num1=3, num2=3){

    return num1 + num2;

}

console.log(addNum(5,6))

***arrow function***

this is very handy and clean things up, it introduced in Es6 i.e Es 2015;

const addNum=(num1=3, num2=3)=>{

    return num1 + num2;

}

console.log(addNum(5,6))

we can also write the code in one line that is if we are not assigning a variable

const addNum=(num1=3, num2=3)=>console.log(num1 + num2)

addNum(5,6)

or

const addNum = (num1, num2) => num1 + num2;

console.log(addNum(5,6)) ;

if you have only one parameter then you don’t need the bracket.

const addNum = num1 => num1 + 5;

console.log(addNum(5)) ;

They have the lexical **(this)** key word as an extra advantage.

**OOP (prototype & Classes)**

We can construct object using the construction function using the prototype and ES6 classes

*Constructor function*

// construction function

function Person(firstName, lastName, dob){

    this.firstName = firstName;

    this.lastName = lastName;

    this.dob = dob;

}

    // initialize object

const person1 = new Person('Nathaniel', 'Nosa', 'April 14');

console.log(person1)

we can pass in the date of bate as a date function instead of string.

// construction function

function Person(firstName, lastName, dob){

    this.firstName = firstName;

    this.lastName = lastName;

    this.dob = new Date(dob);

}

    // initialize object

const person1 = new Person('Nathaniel', 'Nosa', 'April-14-1999');

console.log(person1.dob)

we can add methods which are basically function to the Person objects

// construction function

function Person(firstName, lastName, dob){

    this.firstName = firstName;

    this.lastName = lastName;

    this.dob = new Date(dob);

    this.getMyYear = function(){

        return this.dob.getFullYear();

    }

}

    // initialize object

const person1 = new Person('Nathaniel', 'Nosa', 'April-14-1999');

console.log(person1.getMyYear())

*Prototype*

Person.prototype.getFullName = function(){

    return `${this.firstName} ${this.lastName}`;

}

console.log(person1.getFullName())

*class*

Using the class method is very easy that why most developers called it syntactic sugar.

class Person{

    constructor(firstName, lastName, dob){

    this.firstName = firstName;

    this.lastName = lastName;

    this.dob = new Date(dob);

    }

    getFullName = function(){

            return `${this.firstName} ${this.lastName}`;

    }

}

// initialize a person object

const person1 = new Person('Olamide', 'Falanna', 'May-20-1999');

const person2 = new Person('Nathaniel', 'Nosa', 'April-14-1999');

console.log(person1)

// console.log(person2.dob.getFullYear())

console.log(person1.getFullName())

**DOM Selection**

This means selecting things from the html element and can put them into variable and working with them.

We have single element selectors and multiple element selectors.

In the browser we have what we call the window object, this is the parent object of the browser.

*Single element selectors*

// get element by id

console.log(document.getElementById('form'));

const form = document.getElementById('form');

console.log(form);

// get element by queryselector

console.log(document.querySelector('card'));

const card = document.querySelector('card');

console.log(card)

*Multiple element selector*

// get element by class name

console.log(document.getElementsByClassName('.list-group-item'));

// get element by tag name

console.log(document.getElementsByTagName('li'));

// get element by queryselector all

console.log(document.querySelectorAll('.list-group-item'));

**DOM Manipulation**

Dom actually means the user interface i.e the html point of view. Manipulation means doing basic editing or changes to the DOM.

const ul = document.querySelector('.list-group');

ul.remove()

ul.lastElementChild.remove()

ul.firstElementChild.textContent='Olamide';

ul.children[2].innerText = 'Anthonia';

ul.lastElementChild.innerHTML='<h1>Mr Efe</h1>';

const btn = document.querySelector('.btn');

btn.style.backgroundColor = 'red';

**Event**

These are actions that occurs in the DOM base on certain events, we have several kinds of event, but before we use this we have to add the “addEventListener” to the element that want to process the action. And this addEventListener takes in two parameters (‘the event we want’, function that will be run when the event happens).

btn.addEventListener('click', function(){}) //bad approach

btn.addEventListener('click', aListener)    // recommended

when you are using event it takes in a event parameter e, to prevent the form to submit to a file.

btn.addEventListener('click', (e)=>{})

const btn = document.querySelector('.btn');

btn.addEventListener('click',(e)=>{

    console.log('click');

    e.preventDefault()

})

*Event objects*

Using the e.target we can get more information about the attributes of an html element.

**Basic Form Validation**

*Html view*

<div class="card shadow p-5">

                    <form action="" id="form">

                        <h3>Add User</h3>

                        <!-- message -->

                        <div class="msg"></div>

                        <!-- name -->

                        <div class="form-group mb-3">

                            <label for="name">Name</label>

                            <input type="text" id="name" class="form-control">

                        </div>

                        <!-- email -->

                        <div class="form-group mb-3">

                            <label for="email">Email</label>

                            <input type="text"  id="email" class="form-control">

                        </div>

                        <!-- btn -->

                        <div class="d-grid gap-2 my-3">

                         <button class="btn btn-primary" type="submit"> Submit</button>

                        </div>

                    </form>

                    <!-- users -->

                    <h3>Users</h3>

                    <ul class="list-group" id="users"></ul>

                </div>