**DAY 1 OF REACT (26/08/24)**

Learning core JavaScript concepts

**1. Object/array destructuring** (read more on this)

- Destructuring allows us to extract values from arrays and objects and store them in variables in a convenient way .

we use the "[ ]" for array destructuring while we use "{ }" for object destructuring.

Example.

Const names = [ “David”, “Anita”, “Bola”]

// array destructuring

[firstName, secondName, thirdName] = names

Console.log(firstName)// returns “David”

Let details = {

name: “Jerry”,

age: null,

career: software engineer,

}

{first, second, third}= details

Console.log(second) // returns null

**2. The spread operator**

-The spread operator (...) is a useful feature in JavaScript that allows you to quickly copy all or part of an existing array or object into a new one.

**Example**

const a = ["dog", "cat", "snakes", "goat"]

const b = [...a] // this initializes array b to have the same contents as array a.

**3. Template literal**

Template literals are a new way to define strings in the javascript ES6 module, they are enclosed by backticks(` `) instead of single or double quotes. They are informally called template strings and can be used for string interpolation.

**Example**

Let name;

Name = `Egemonye Jeremiah`;

fullDetail = `My name is ${Name}` //outputs “My name is Egemonye Jeremiah”

**4.Ternary operator**

Ternary operators in JavaScript are a form of syntactic sugar for quickly writing if-else statements. They are called ternary because they involve three operands.

**Syntax**

condition ? exprIfTrue : exprIfFalse.

**Example**

let a = 5;

a>6?console.log(“a is greater than 6”): console.log(“a is not greater than 6”);

// returns “a is not greater than 6”

**5. Arrow functions**

This is a new way of writing single line fuctions, it was introduced into javascript ES6.

**Syntax**

(args)=> //code to return;

Example

Const calculate(a,b)=> a+b;

Calculate(4,5) // returns 9

The example above is for a function with a single return statement. For function which would occupy multiple lines of code we would use the syntax

Const calculate =(a,b)=>{

//input codes here

//manually input the return statement

}

**NB:** Function expression and function declaration

**DAY 2 OF REACT**

**6. Short circuiting and logical operators**

In JavaScript, "short-circuiting" is a behavior in logical && (AND) and || (OR) operators. If the first operand is enough to determine the result, the second operand is not evaluated. For &&, if the first operand is falsy, it returns the first operand; otherwise, it returns the second operand. For ||, if the first operand is truthy, it returns the first operand; otherwise, it returns the second operand.

Syntax

//for the logical and operator

Let a = 500;

a>100 && console.log(“a is greater than 100”)

i.e for the logical **and** operator if the code on the left is true, it executes the code on the right

//for the logical **or** operator

Let b = 300;

b>200 || console.log(“”)

i.e for the logical or operator, if the code on the left is true, it executes and return that code, meanwhile if the code is false it returns the code on the right.

**7. Optional chaining**

The **optional chaining (?.)** operator accesses an object's property or calls a function. If the object accessed or function called using this operator is [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) or [null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/null), the expression short circuits and evaluates to [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) instead of throwing an error.

Syntax

Let food = {

{

Id: 1,

Name: “rice”

}  
 {

Id: 2,

Name: “random”

}

}

Let foodName = food?.name;

//the optional chaining operator above asks the question ‘if the food object exists’, access the name property

**NB:  
8. The nullish coalescing operator**

The nullish coalescing operator (??) in JavaScript is a logical operator that returns the right-hand side operand if the left-hand side operand is null or undefined.

**9. The array map method**

The array map method is used to loop through an array and it executes a callback function for each element in the array.

Syntax

Let names = [“me”, “you”, “we”]

Let newNames = names.map((name)=> name = “Jerry”);

//here it replaces all the elements in the array with string “Jerry”.

**10. The array filter method**

filter out an array based on meeting a certain condition

**11. The array sort method**

Unlike the map and filter method, the sort method mutates the original array based on meeting a certain condition.

NB:   
**If we want to add objects to an array of its type, we can make use of the spread operator.**

Syntax

Let data=[{title:”movie”, id:2}]

Let data=[…data, {more…..}]

**If we want to delete objects from the array we can use a filter method.**

**If we want to update the contents of the array we can make use of a map method.**

**Asynchronous JavaScript**

Promises  
-fetching API’s data using async await

Syntax

Async function data(){

Let special = await fetch(‘API endpoint here’)

Console.log(special.json());

}

data();

React

Components, these are the building blocks of a user interface in react

(component tree)

**Day 3 of react**

-Built my first project

-made use of inline component styling,

- made use of props to pass on data from a parent to a child component

-Learnt core JSX syntax

**Rendering lists**

-Using the array map method to render lists(*super cool*😅) tho need to revisit this when revising.

**Conditional rendering**

Conditional rendering renders a piece of UI upon meeting a certain condition

1. Conditional rendering using the “&&” operator (making reference to short circuiting)
2. Conditional rendering using the ternary operator

-an advantage of using the ternary operator over the and operator is we can render an alternative component if the condition isn’t met unlike the and operator if the condition is not met, it returns null and cannot render an alternative UI.

3. Rendering using return statements: if a condition is met we can return a particular piece of UI this is best practiced when trying to render individual components, not a piece of JSX code

**Destructuring Props**

We destructure props by passing in the data to the in the child component in curly braces rather than declaring passing props as an argument In the parent component and accessing in the child component using “props.----“. The syntax will explain better

Syntax

**Props without destructuring**

Function Head(){

….

….

< Tail name={name}/>

}

Function Tail(props){

….

…..

<div>{props.name}</div>

}

**Props with destructuring**

Function Head(){

….

….

< Tail name={name}/>

}

Function Tail({name}){

….

…..

<div>{name}</div>

}

**NB:** After destructuring the props from the parent component, we can access it in any part of the child component without calling the “props” object.

**React fragment**

-React fragment is a feature in react that allows you to return multiple elements from a react component by allowing you to group a list of children without adding extra nodes to the DOM

**Setting classes and texts conditionally**

Syntax

Let allowed = true;

<div className= {`${allowed? ’class to render if true’:’class to render if false’}`}>{allowed?’hii’:’bye’}</div>

**Event handling in react**

Handling events in react, we use camel case syntax on the element

E.g

<div onClick={handle}></div?

**State and state management**

A state in react is a built in object that allows component to manage dynamic data and re-render the component when the data changes.

**Controlled elements**

Controlled elements in react refers to form elements whose value is controlled by react state. This means that the form elements derive their values from state and notify react of changes through event handlers.

**State vs Props**

**NB: Read on array methods especially the .from() method**

**State lifting**

When there are two or more sibling components in need of a common state we look through our react tree to find the component higher up the tree that can pass down the states to all the components in need of it.

**Derived state**

Practice project

**Build a to-do-list app**

-learnt how to store and retrieve data from local storage

-Applied the concept of state lifting

-Made use of the array sort method to rearrange components in the list

-Learnt how to build and accordion