**name:-omatiga Joshua iko-ojo**

**matric no :- 23/2000**

**department :-information technology (group c )**

**Order Functions and Closures in JavaScript**

1. **Higher-Order Functions**

A higher-order function is basically a function that can take other functions as inputs or even return a function. This can be super helpful because it lets you write reusable code that you can adjust as needed.

Some Examples

- Using map :

javascript

const numbers = [1, 2, 3, 4];

const doubled = numbers.map(num => num \* 2);

console.log(doubled); // [2, 4, 6, 8]

Here, `map` is a higher-order function because it takes a function (`num => num \* 2`) as an argument. This function is applied to each item in the array, which is why all the numbers get doubled.

- Returning a Function:

javascript

function createMultiplier(factor) {

return function(number) {

return number \* factor;

};

}

const double = createMultiplier(2);

console.log(double(5)); // 10

In this example, `createMultiplier` returns a function that multiplies a number by the `factor` you choose. This way, you can create a function that doubles, triples, etc., depending on what `factor` is.

2. **Closures**

A closure is when a function keeps access to variables from its original scope, even if that function is used outside of that scope. Closures are really useful for storing data that you don’t want anyone else messing with.

Example of a Closure

javascript

function makeCounter() {

let count = 0;

return function() {

count++;

return count;

};

}

const counter = makeCounter();

console.log(counter()); // 1

console.log(counter()); // 2

console.log(counter()); // 3

```

In this example, `count` is only available inside the `makeCounter` function, but the function that gets returned (the `counter` function) can still see and update `count`. So, every time you call `counter()`, it goes up by 1.

Why Are Closures Useful?

Closures let you:

- Keep Data Private: You can create variables that only certain functions can see and change.

- Keep Track of Values : A closure lets a function remember certain values, even after it’s done running.

How They Work Together

Higher-order functions and closures are even better when you use them together. Check out this example:

javascript

function createCounter(start) {

let count = start;

return {

increment: () => ++count,

decrement: () => --count,

};

}

const counter = createCounter(10);

console.log(counter.increment()); // 11

console.log(counter.increment()); // 12

console.log(counter.decrement()); // 11

Here:

- `createCounter` is a higher-order function because it returns an object with two functions inside it.

- The `increment` and `decrement` functions are closures because they can still use the `count` variable, even though `count` isn’t in their scope anymore.