Q.1 Are Higher Order functions and Call back functions the same ? If not, briefly explain about both functions.
Ans: Though they are not precisely the same, higher-order functions and callback functions are related ideas in programming, especially in functional programming paradigms like JavaScript.

Higher-order functions: A higher-order function is one that either returns a function as its result or accepts one or more functions as arguments.
Stated otherwise, a higher-order function has the ability to manipulate functions in the same manner as ordinary functions do with data.
Callback features:

A callback function is a function that is called within the body of another function after being supplied as an argument to that other function.
In asynchronous programming, callback functions are frequently utilised. In this scenario, a function receives a callback function as an argument and calls it after the function has finished running.

Q.2 Is filter a Higher Order function in Javascript ? If yes, why ?
Ans: In JavaScript, the filter function is in fact a higher-order function.

A function that returns a function as its result or accepts one or more functions as arguments is known as a higher-order function. Because filter accepts a callback function as a parameter, it satisfies this definition.

The filter method in JavaScript generates a new array containing every element that passes the test carried out by the supplied callback function. Every element in the array is subjected to the callback function, and the elements for which the callback returns true are included into the newly created array.

Q.3 Give an example of a Higher Order function and a call back function used in the same program.
Ans:
Sure, here's an example of a higher-order function and a callback function used in the same JavaScript program:

```
// Higher-order function: map
function multiplyByTwo(array, callback) {
   return array.map(callback);
}

// Callback function: square
function square(number) {
   return number * number;
}

// Array of numbers
const numbers = [1, 2, 3, 4, 5];

// Using the higher-order function with the callback function
const squaredNumbers = multiplyByTwo(numbers, square);
```

console.log(squaredNumbers); // Output: [1, 4, 9, 16, 25]

In this instance:

Because multiply By Two accepts an array and a callback function as inputs, it is a higher-order function. Next, using a map, it applies the callback function to every element in the array.
Since square is called within map for each element of the array and is supplied as an argument to multiply By Two, it is a callback function.

Using the square callback function, the programme takes an array of numbers and squares each value in the array using multiply By Two. An array of squared numbers is the outcome. This example shows how callback functions and higher-order functions can be combined to create code that is more reusable and versatile.

Q. 4 Carefully check the example below:
a) What will be the output of this program?
b) Which function is a Higher Order function here?

```
const names= ['John', 'Tina','Kale','Max']
function useFunction(arr,fn){
for(let i=0; i<arr.length; i++){
fn(arr[I]);
}
}
function argFn (name){
console.log("Hello " + name );
}
useFunction(names,argFn);
```

Ans:

There's a small typo in your code where `arr[I]` should be `arr[i]` (lowercase "i"). Here's the corrected version:

```
const names = ['John', 'Tina', 'Kale', 'Max'];

function useFunction(arr, fn) {
  for (let i = 0; i < arr.length; i++) {
    fn(arr[i]);
  }
}

function argFn(name) {
  console.log("Hello " + name);
}

useFunction(names, argFn);
```

Now, let's address your questions:

a) The output of this program will be:

```
Hello John

Hello Tina

Hello Kale

Hello Max
```

The `useFunction` function iterates over the `names` array and applies the `argFn` function to each element of the array. The `argFn` function simply logs "Hello " followed by the name passed to it. Therefore, for each name in the `names` array, "Hello" followed by the name will be logged to the console.

b) In this code, `useFunction` is the higher-order function. It takes an array (`arr`) and a function (`fn`) as arguments. Inside `useFunction`, it iterates over each element of the array and applies the provided function (`fn`) to each element. Thus, `useFunction` is the higher-order function because it takes a function as one of its arguments and applies it to each element of the array.