Q1 .
Cookie are used to store small amount of data such as session variables in the web browser.
Local storage are used to store large amount of data such as key-value pair in the web browser and it remains after the web browser closed.
Session storage are used to store large amount of data in the web browser but the session storage data will remain only for the current browser session.

Q2. Output of the given code:

```
5
5
5
5
5
```

Explaination :

1. For every iteration of loop, the value of i will be 0,1,2,3,4 respectively.

2. For each iteration, setTimeout function will be called and scheduled a delay of 100 ms for the function to get executed.

3. And make a entry in the callback queue.

4. Once the for loop ends, the event loop start pushing every element of callback queue to call stack in FIFO order. And then execute it.

5. Now, the setTimeOut function will get executed and will show i=5 because at the end of iteration, i will become 5 (clousre property).

Q3. Sharding in MongoDB is a data distribution technique used to horizontally scale databases across multiple servers (or nodes) to handle large amounts of data and high read/write workloads

Sharding works by dividing the data into smaller, more manageable chunks called shards. Each shard is a separate and independent MongoDB replica set that can be distributed across different machines or clusters.

Q4 . Promise chaining is a technique used in JavaScript to chain multiple asynchronous operations together using Promises. This allows you to execute a series of asynchronous tasks one after another, making the code more readable and easier to maintain.

Q5. Higher-Order Component (HOC) is an advanced pattern used for reusing component logic, adding functionality, or modifying component behavior.

Working: It takes a component as an argument and returns a new enhanced component with additional props and behavior. Steps Involved are:

1. Firstly, it create the HOC Function
2. Modify the WrappedComponent
3. Returned the enhanced component
4. Use the enhanced component

Q6. Callback hell is a situation in asynchronous programming where nested and deeply indented callback functions become difficult to read and manage. It occurs when multiple asynchronous operations are executed one after another, and each operation relies on the result of the previous operation.

Different ways to solve callback hell are:

1. Promise
2. Async/Await
3. Named function

Q7.

```
const arr = [1, 2, 3, 4, 5];

const reverseArray = arr.reduce((accumulatorArray, currentArray) => {

  accumulatorArray.unshift(currentArray); // Unshift function add the current
element to the beginning of the accumulator array.

  return accumulatorArray;

}, []);

console.log(reverseArray);
```

Github Link : [**PPT_Assignment/IBI Group Test at main · nehalgarg7/PPT_Assignment (github.com)**]

Q8. Output : 1 4 3 2

Explanation :

1. First starting statement of function will get executed --> print 1
2. then setTimeOut function will be called and scheduled a delay of 1000 ms for the function to get executed.
3. then another setTimeout function will be called and scheduled a delay of 0 ms for the function to get executed.
4. then last statement will get executed and print ---> 4.
5. Once the code ends...then setTimeout function will wait in the callback Queue in FIFOorder for further execution.
6. Event loop will check for the call stack(currently empty) and then push the setTimeOut function to callstack for the further execution one by one in FIFO order.

Q9. Output : array 1: length=5 last=j,o,n,e,s

array 2: length=5 last=j,o,n,e,s

Explaination : After every line of code :

1. arr1 is ["j", "o", "h", "n"]

2. arr1 is now ["n", "h", "o", "j"] and arr2 points to the same array as arr1

3. arr3 is ["j", "o", "n", "e", "s"]

4. Since arr2 is a reference to the same array as arr1, this pushes arr3 into the same array, modifying it.

5. arr1 is the modified array after arr2.push(arr3), so it contains ["n", "h", "o", "j", ["j", "o", "n", "e", "s"]]. The length is 5, and the last element is the array ["j", "o", "n", "e", "s"].

6. arr2 points to the same array as arr1, so the result is the same as the previous

Q10. Output: 1,4,2,3

Explaination:

1. first console log print ---> 1

2. setTimeOut function will call and wait for 0 ms for execution .

3. another setTimeOut function will call and wait for 100 ms for execution .

4. Last console log print ---> 4

5. Then event loop take the first setTimeout function from callback queue to call stack and then execute it. And same will happen with other setTimeout function.

Q11. Code:

```
const numbers = [1, 2, 3, 4, 5];

const result = numbers.reduce((acc, num) => {

  if (num % 2 === 0) {

    acc.evens.push(num);

  } else {

    acc.odds.push(num);

  }

  return acc;

}, { evens: [], odds: [] }); // Initialize accumulator array as an object with evens and odds properties.


console.log(result);
```

Output : { evens: [2, 4], odds: [1, 3, 5] }

Explaination : With this modification, the reduce function will correctly initialize the accumulator (acc) as an object with evens and odds properties. The evens array will contain even numbers, and the odds array will contain odd numbers from the numbers array.

Github Link : PPT_Assignment/IBI Group Test at main · nehalgarg7/PPT_Assignment (github.com)