**Q1.What are microtasks? What is a microtask queue? What is their role in Promises and how are they different from callbacks?**

**Ans:** JavaScript is asynchronous and so does the REACT. The secret behind this asynchronous nature is one smart idea termed as EVENT LOOP.JS and REACT JS executes the operation in synchronous fashion but with the help of few smart data structures they give feeling of multithreading and those data structures **are CALL STACK,TASK QUEUE**(also known as Macrotasks) and **JOB QUEUE(**also named as **microtasks queue).**So Microtasks queue is a data structures (FIFO type) that holds the callbacks of promises that are ready to be executed. Microtask is small javascript code (function) which gets executed after the function or any program that might have created it and only if execution stack i.e call stack is empty.

The major difference both is that when using callback , a callback is normally passed into a function that would get called upon completion to get the desired result whereas in promises ,callbacks are attached on the returned promise object. To gain remedy for the problem of callback hell or pyramid of DOM there introduced the concept of promises.

Then webapi callback is enqueued to task queue whereas the promise callback is enqueued to job queue(microtasks queue).

**Q2. Explain with examples how private, protected variables can be implemented in classes and how can they be used in subclasses?**

Ans: In Javascript the private members are prefixed using underscore(\_) so that everybody knows they are private but that doesn’t make any variable private.

In javascript we can implement the private variable using closures.

For eg:

function incrVal() {

var count = 0;

return {

incr : function () {

return count++;

}

}

}

var a = new incrVal();

var b = new incrVal()

console.log(a.incr()); // 0

console.log(a.incr()); // 1

console.log(b.incr()); // 0

console.log(b.incr()); // 1

console.log(a.incr()); // 2

console.log(b.incr()); // 2

console.log(count);//undefined

Here a and b maintain their own private count variable.

* **Now in order to implement protected variable we can use getter property of class.**

For eg:

class EmpName {

\_name;

constructor(name) {

this.\_name = name;

}

get name() {

return this.\_name;

}

}

let empName = new EmpName("John");

console.log(`My name is ${empName.name}`); // My name is John

empName.name = "Jane"; // Cannot assign to 'name' because it is a read-only property

**And Using them in subclass can also be done.**

For eg:

function Incr() {

let count = 22

return {

getCount: () => count,

setCount: input => {

count = input

}

}

}

function IncrExtend() {

let parent = new Incr(),

{ getCount, setCount } = parent

return Object.assign(parent, {

getCountDouble: () => getCount() \* 2

})

}

let extendedIncrObj = new IncrExtend()

console.log(extendedIncrObj.getCount()) // 22

console.log(extendedIncrObj.getCountDouble()) // 44

extendedIncrObj.setCount(11)

console.log(extendedIncrObj.getCount()) // 11

console.log(extendedIncrObj.getCountDouble()) // 22