**Question number 1**

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

**Ans:**

Microtasks are usually those scheduled tasks that should happen straight after the currently executing script, such as reacting to a batch of actions, or to make something async without taking the penalty of a whole new task. The microtask queue is processed after callbacks as long as no other JavaScript is mid-execution, and at the end of each task. Any additional microtasks queued during microtasks are added to the end of the queue and also processed.

Examples would be promise callbacks, web API calls and mutation observer callbacks.

All the callback functions coming through Promises and Mutation Observer will go inside the Microtask Queue. In the case of .fetch(), the callback function gets to the Microtask Queue. They have higher priority than callback queue which gets functions from setTimeouts.

Once a promise settles, or if it has already settled, it queues a microtask for its reactionary callbacks. This ensures promise callbacks are async even if the promise has already settled. So, calling .then() against a settled promise immediately queues a microtask.

### Question number 2

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

**Ans:**

Private variables:

We can create them using closures and \_notation.

**Using closure:**

function carSpeed() {

var speed = 0;

return {

accelerate: function () {

return speed++;

}

}

}

var car = new carSpeed();

console.log(car.accelerate()); // 0

console.log(car.accelerate()); // 1

console.log(speed); // speed is not defined

**Using # notation:**

class myObject {

\_meaningOfLife;

constructor(name) {

this.\_meaningOfLife = 12;

}

returnMeaningOfLife() {

return this.\_meaningOfLife;

}

}

const obj = new myObject("Himanshu");

console.log(obj.returnMeaningOfLife()); // 12

console.log(obj["#meaningOfLife"]); // undefined

We can use inheritance to access the protected variables in subclasses as private members can’t be used outside the main class. Passing the variables as arguments in the subclasses will help accessing the same.