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

Asynchronous tasks need proper management. For that, the ECMA standard specifies an internal queue Promise Jobs more often referred to as the “microtask queue” (V8 term).

Microtask queue or Job queue for all async operations with higher priority.

Like Promises,process.nextTick,object.observe,MutationObserver.

It a queue to hold the above-mentioned task(until call stack is empty) and give them high priority by sending them to the call stack for execution, when ever it is empty. Event loop will be continuously checking for call stack and queue.

In case of callbacks we pass the callback function as parameter to another function. And that will be called upon completion of the async task.

In Promise, we attach the call-backs on the returned promise object.

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

**Private Variable:**

Private variable are only accessible within the class that instantiated the object.  
Can be included in the class by prefixing the variable with “#” which is introduced as part of ES2019/10

class StudentModal {

  #empFirstName;

  #empLastName;

  getUserFullName(name) {

    this.#empFirstName = name.firstName;

    this.#empLastName = name.lastName;

    return this.#empFirstName + ' ' + this.#empLastName;

  }

}

const student1 = new StudentModal();

console.log(

  student1.getUserFullName({ firstName: 'Shruthi', lastName: 'patel' })

);

console.log(student1.firstName);// undefined. Since private cannot be accessed outside the class

**Protected Variables:**

Protected variables allows a little more access than private members but a lot less than the public. A protected member is accessible within the class and any object that inherits from it. A protected value is shared across all layers of the prototype chain. It is not accessible by anybody else.

class NameGenerator {

  \_name;

  constructor(name) {

      this.\_name = name;

  }

  get name() {

      return this.\_name;

  }

}

let nameGenerator = new NameGenerator("Shruthi");

console.log(`My name is ${nameGenerator.name}`); // My name is Shruthi

nameGenerator.name = "Pooja"; // Here we get the error Error: Cannot set property name of #<NameGenerator> which has only a getter