1. The first question you will have to answer is explaining JS event loop, clarifying the synchronous and asynchronous parts, queues, priority, with examples. Write your answer as if it were for a real interview.

Answer:

JS event loop is a continuously executing single-threaded while loop running on main thread of JS which is responsible for listening to the different events and actions queued together and executing them based on FIFO principle.
Event Loop comes together with other parts to built up a runtime model for the JS which consists of JS Engine, Web APIs and Queue. All these parts work together to execute a JS code.
In JS synchronous code are always executed before asynchronous code. So, at first with the help of JS Engine all of the synchronous code are executed from a stack which is responsible for execution. Along with this whenever the asynchronous code is found on the way they are delegated to the Web APIs which are later queued and made ready to execute.
Here, the event loop looks the execution stack whether it is empty, if yes then the queued tasks are transferred to the stack one by one, and execution is done. There are two task queues;

a. Macro-Task Queue or Callback Queue: This queue stores timer functions and normal event handlers. Event loop only looks for this queue.
b. Micro-Task Queue: This queue stores the asynchronous functions which returns promise. It is bundled with V8 engine and also has more priority than macro-task queue.

Example:

```
console.log("First line"); // 1 -> synchronous code
const promise = new Promise((resolve, reject) => {
  console.log("Inside promise"); // 2 -> synchronous code
  setTimeout(() => { // 6 -> asynchronous code goes to macro-task queue
    resolve("Inside promise inside set timeout");
  }, 1000);
});
setTimeout(() => {
  console.log("Inside set timeout"); // 7 -> asynchronous code goes to macro-task queue
}, 1000);

const promise2 = new Promise((resolve, reject) => {
  resolve("Inside second promise"); // 5 -> asynchronous code goes to micro-task queue
});

console.log("Middle of the code"); // 3 -> synchronous code
promise2.then((result) => {
  console.log(result);
});
promise.then((result) => {
  console.log(result);
});
console.log("Last Line"); // 4 -> synchronous code
```

2. The follow up question is how may we convert a sync operation/function to become asynchronous?

Answer:

We may convert a sync operation/function to become asynchronous with the help of following ways;

1. Using JavaScript timer functions like setTimeout or setInterval if we want to execute the code once or periodically after some period of time.
2. Using promise or async/await to convert the function to be asynchronous and handle its resolve and reject portion using then, catch or try, catch.