1. The JS event loop in the browser, clarify the synchronous and asynchronous parts, queues, and priority.

   The event loop is Javascript is the browser's runtime environment that handle the execution of Javascript code.

   Synchronous means one process is running at a time and blocking other operations until the current operation is complete.

   Asynchronous means more than one process running simultaneously. Asynchronous delegates the operation to someone else and consume the result. In javascript, asynchronous operations can be used callbacks, promise and async-await mechanisms. Asynchronous operations includes fetching data from the server, download files, images and so forth.

   Queue

   Queue is waiting area for functions, that is managed by the event loop and includes two main tasks – Macro Task and Micro Task.

   I/O operations like DOM events, setTimeout, setImmediate, UI events will go through the macro queue. And, operations like fetch(), resolve and reject from the Promise will go through the micro task queue.

   Priority

   The event loop always prioritizes the micro task queue over macro task queue. The event loop checks the micro task queue when each task is finished, runs every available micro task until the queue is empty, and then moves on to the next macro task queue.

```
(function(){
    setTimeout(()=> {console.log("setTimeout Result")});
})();

console.log("starts");

const promise = new Promise((reslove, reject) => {
    console.log("Promise starts");
    reslove("Promise Result");
});
promise.then(console.log);
```

```
console.log("end");
```

In the above example, the function that calls setTimeout() Web API will add to the macro task queue and resolve() part in the Promise will add to the micro task queue. The synchronous parts 'starts', 'Promise starts' and 'end' will print in console first. After that, the operation from micro task queue, 'Promise Result' will print. In the end, the output from the setTimeout function 'setTimeout Result' will print. So the output will be:

Starts
Promise starts
End
Promise Result
setTimeout Result

2. How may we convert a synchronous function to become asynchronous?

   We can convert a synchronous function to become asynchronous by using Web APIs like queueMicrotask() and setTimeout() which transfer javascript jobs to other threads.

   queueMicrotask() is a global function which is exposed on the window which stores callback function in microtask queue.

```
function performMicrotask(){
    console.log("executing microtask");
}

function scheduleMicrotask(){
    queueMicrotask(performMicrotask);
}

console.log("start");

scheduleMicrotask();

console.log("end");
```

In the above example, performMicrotask() function has simple console.log function and scheduleMicrotask() function call the above function with queueMicrotask() to schedule the execution as the microtask. So, after

synchronous functions complete, ‘start’ and ‘end’ outputs in the console, the scheduleMicrotask() function will execute and the output is:

start
snd
executing microtask

References: https://developer.mozilla.org/en-US/docs/Web/API/queueMicrotask

setTimeout(callback, delay) function stores the callback function in macro queue temporarily. After ‘delay’ specific time milliseconds, the callback function will put in V8 stack when the stack is empty. And the callback function will be executed.

The V8 stack is empty when all synchronous codes are executed.

Reference: Lecture Slide

```
function performSettimeout(){
    console.log("executing settimeout");
}

console.log("start");

setTimeout(performSettimeout, 1000);

console.log("end");
```

In the above example, performSettimeout() function has simple console.log function. After synchronous functions complete, ‘start’ and ‘end’ outputs in the console, the setTimeout function will execute after one second and the output is:

start
snd
executing settimeout

3. What is the advantage of an asynchronous function? And what problem does it resolve? Why you may need to create one?

   The advantage of an asynchronous function is increasing the user-experience, performance and responsiveness as it becomes non-blocking execution. Non-blocking execution means delegate long-running operation to worker thread,

ensures that the program remains responsive and can handle multiple thread concurrently.

When we have to request data from the server and need to wait the response and make some operations according to the responses, the function needs to be asynchronous.