1. How many callbacks queues in Node.js and what are they? Give examples to explain how different callbacks are enqueued in different queues.

Ans:

Node.js has all together six callbacks queues.The are

i. **nextTick**

Example:

```
console.log("Start");
process.nextTick(() => {
   console.log("Next Tick");
});
Promise.resolve().then(() => {
   console.log("Promise Microtask");
});
console.log("End");
```

In this example, we have both `process.nextTick()` and a promise. Here's what happens:

1. "Start" is logged to the console.
2. The callback scheduled with process.nextTick() is enqueued to the next tick queue.
3. The promise is resolved immediately (Promise.resolve()), and itis added to the microtask queue.
4. "End" is logged to the console.
5. At the end of the current event loop iteration, microtasks are processed before continuing to the next tick. So, "Promise Microtask" is logged first.
6. After processing all microtasks, the event loop moves on to the next tick, executing the callback scheduled with process.nextTick(). So, "Next Tick" is logged last.

Output

```
Start
End
Promise Microtask
Next Tick
```

ii. Promise: Explained in (i)

iii. Timer:

```
process.nextTick(() => console.log("this is process.nextTick 1"));
setTimeout(() => {
   console.log("this is setTimeout 2");
   process.nextTick(() =>
      console.log("this is inner nextTick inside setTimeout")
   );
}, 0);

Promise.resolve().then(() => {
   console.log("Promise Microtask");
});

console.log("Start");
process.nextTick(() => {
   console.log("Next Tick");
});
```

iv.

- Callbacks in the Microtask Queues are executed before callbacks in the Timer Queue
- Callbacks in microtask queues are executed in between the execution of callbacks in the timer queue.
- So, the Output will be as follows:

```
Start
this is process.nextTick 1
Next Tick
Promise Microtask
this is setTimeout 2
this is inner nextTick inside setTimeout
```

v. I/O

Example:

```
const fs = require('fs');
fs.readFile('hello.txt', () => console.log('this is readFile 1'));
new Promise(resolve => resolve('Hi')).then(() => console.log("this is
Promise.resolve 1"));
process.nextTick(() => console.log("this is process.nextTick 1"));
```

- Callbacks in the microtask queue are executed before callbacks in the I/O queue.
- So, the output of above code will be

```
this is process.nextTick 1
this is Promise.resolve 1
this is readFile 1
```

vi. Check

Example:

```
const fs = require("fs");
fs.readFile('hello.txt', () => {
    console.log("this is readFile 1");
    setImmediate(() => console.log("this is setImmediate 1"));
    process.nextTick(() =>
        console.log("this is inner process.nextTick inside readFile")
    );
    Promise.resolve().then(() =>
        console.log("this is inner Promise.resolve inside readFile")
    );
});
process.nextTick(() => console.log("this is process.nextTick 1"));
Promise.resolve().then(() => console.log("this is Promise.resolve 1"));
setTimeout(() => console.log("this is setTimeout 1"), 0);
for (let i = 0; i < 2000000000; i++) { }
```

- Check queue callbacks are executed after microtask queues callbacks, timer queue callbacks and I/O queue callbacks are executed.
- So, the output will be as follows:

```
this is process.nextTick 1
this is Promise.resolve 1
this is setTimeout 1
this is readFile 1
this is inner process.nextTick inside readFile
this is inner Promise.resolve inside readFile
this is setImmediate 1
```

vii. Close

Example:

```
const fs = require("fs");
const readableStream = fs.createReadStream('hello.txt');
readableStream.close();
readableStream.on("close", () => {
console.log("this is from readableStream close event callback");
});
setImmediate(() => console.log("this is setImmediate 1"));
setTimeout(() => console.log("this is setTimeout 1"), 0);
Promise.resolve().then(() => console.log("this is Promise.resolve 1"));
process.nextTick(() => console.log("this is process.nextTick 1"));
```

- Close queue callbacks are executed after all other queue callbacks in a given iteration of the event loop.
- So, the output will be as follows:

```
this is process.nextTick 1
this is Promise.resolve 1
this is setTimeout 1
this is setImmediate 1
this is from readableStream close event callback
```