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

There are 8 callbacks queues.

* 1. Timers

Executes callbacks scheduled by setTimeout() and setInterval().

* 1. I/O callbacks

This is pending callbacks and executes I/O callbacks deferred to the next loop iteration.

* 1. Idle, prepare

Only used internally

* 1. Poll

It retrieves new I/O events; executes I/O related callbacks

* 1. Check

It invokes setImmediate() callbacks.

* 1. Close callbacks

Callbacks from the close event like close connections, close socket.

* 1. nextTick

callbacks from process.nextTick()

* 1. Promise

Callbacks from promise’s resolve and reject

1. What's the output in the code below? Try to understand how it works as we explained in the class

console.log('start');

setTimeout(() => console.log('timeout 1'), 0);

setTimeout(() => {

    console.log('timeout 2')

    process.nextTick(() => console.log('nextTick 3'));

}, 0);

setTimeout(() => console.log('timeout 3'), 0);

new Promise((resolve, reject) => resolve('Hello')).then(() => console.log('Promise...1'));

new Promise((resolve, reject) => resolve('Hello')).then(() => console.log('Promise...2'));

process.nextTick(() =>  console.log('nextTick 1'));

process.nextTick(() =>  console.log('nextTick 2'));

console.log('end');

output:

start

end

nextTick 1

nextTick 2

Promise…1

Promise…2

timeout 1

timeout 2

nextTick 3

timeout 3

3. What's the output of the code below? Try to understand how it works as we explained in the class.

const fs = require('fs');

fs.readFile('hello.txt', () => {

    console.log('readFile....');

});

setTimeout(() => console.log('timeout'), 0);

setImmediate(() => console.log('Immediate'));

output:

timeout

Immediate

readFile….

4. What's the output of the code below when running in Node.js? If for the same code running in browser, what's the output? Why is the output different?

var message = 'Hello World';

function logMessage() {

    console.log(this.message);

}

logMessage();

The output is ‘undefined’ in Node.js but ‘Hello World’ in browser.

The difference in output is due to the default context in which the function is executed. In Node.js, the default context is the global object (global), while in a browser, it's typically the window object. In Node.js, ’this’ object is empty because all .js files are wrapped into function.

5. What will happen when execute the code below? If there's error, what's the error and what cause the error? If no error, what's the output in the console? Assume pattern1.js and app.js are in the same folder

// pattern1.js

module.exports.getName = function () {

    console.log('Josh Edward');

};

// app.js

const getName = require('./pattern1');

getName();

**output will be “getName is not a function”.**

**To fix this, we modify ‘getName.getName()’in app.js**

**Or**

**‘module.exports = function (){…}**

6. What will happen when execute the code below? If there's error, what's the error and what cause the error? If no error, what's the output in the console? Assume pattern1.js and app.js are in the same folder

// pattern2.js

exports.getFirstname = function () {

    console.log('Josh');

};

exports = {

    getLastname: function(){

        console.log('Edward');

    }

}

module.exports = function (){

    console.log('Josh Edward');

}

// app.js

const getName = require('./pattern2');

getName();

**output is ‘Josh Edward’ because return value is functon.**

7.  What will happen when execute the code below? If there's error, what's the error and what cause the error? If no error, what's the output in the console? Assume pattern1.js and app.js are in the same folder

// pattern1.js

exports.getFirstname = function () {

    console.log('Josh');

};

exports = {

    getLastname: function(){

        console.log('Edward');

    }

}

module.exports.getFullname = function (){

    console.log('Josh Edward');

}

// app.js

const {getFullname} = require('./pattern2');

getFullname();

**output is ‘Josh Edward’ because return object with getFullname attribute.**

8. Review IIFE, bind, apply, call methods first, what's the value of result in the console?

const result = (function(exports, module){

    exports = module.exports;

    exports.firstname = 'John';

    module.exports.lastname = 'Smith';

    exports = {

        getFullName: function(){

            console.log('John Smith')

        }

    }

    return module.exports;

}).apply(null, [null, {exports: {}}]);

console.log(result);

**The output is { firstname: 'John', lastname: 'Smith' }**