Assessment : 10
Node JS

Section I [MCQ Questions]: 1 Marks Each

1. Is Node.js Multi Threaded Externally?

- A) True
- B) False

Answer: B) False
Node.js is primarily single-threaded externally, using an event-driven, non-blocking I/O model.

2. In Node Process is _____?

- A) Global Object
- B) Local Object
- C) Instance Scope
- D) NA

Answer: A) Global Object
The `process` object is a global object that provides information and control over the current Node.js process.

3. Which Function is used to Include Modules in Node.js?

- A) include()
- B) require()
- C) attach()
- D) import

Answer: B) require()
The `require()` function is used to include modules in Node.js.

4. What is a Callback?

- a) Callback is an asynchronous equivalent for a function.
- b) Callback is a technique in which a method calls back the caller method.
- c) Both of the above.
- d) None of the above.

Answer: a) Callback is an asynchronous equivalent for a function.
It refers to a function passed as an argument to be executed after another function completes asynchronously.

5. Simple or complex functionality organized in a single or multiple JavaScript files which can be reused throughout your Node.js application is called _____.

- a) Module
- b) Library
- c) Package
- d) Function

Answer: a) Module
Modules are reusable units of functionality organized in files.

6. Which of the following module is required for path specific operations?

- a) os module
- b) fs module
- c) path module
- d) None of the above

Answer: c) path module
The `path` module helps work with file and directory paths.

7. What is package.json?

- a) It is a plain JSON (JavaScript Object Notation) text file which contains all metadata information about Node.js Project or application.
- b) This file should be present in the root directory of every Node.js Package or Module to describe its metadata in JSON format.
- c) The file is named as “package” because Node.js platform treats every feature as a separate component. Node.js calls these as Package or Module.
- d) All of the Above

Answer: d) All of the Above
`package.json` contains metadata for the Node.js project, is placed in the root directory, and describes packages or modules.

8. Route paths, in combination with a request method, define the endpoints at which requests can be made. Which of the following are valid forms of route path?

- a) Strings
- b) String patterns
- c) Regular expressions
- d) All of the Above

Answer: d) All of the Above
Express routes can be defined using strings, patterns, or regex.

9. What are core features of Express framework?

- a) Allows to set up middlewares to respond to HTTP Requests
- b) Defines a routing table which works as per HTTP Method and URL
- c) Dynamically render HTML Pages
- d) All of the above

Answer: d) All of the above
Express supports middleware, flexible routing, and dynamic HTML rendering.

10. How to Connect to MongoDB from Node?

- a) mongoClient.connect(params)
- b) mongoClient.obtainConnection()
- c) mongoClient.geConnection()
- d) None of the above

Answer: a) mongoClient.connect(params)
The `connect()` method of MongoClient is used to establish a connection in Node.js.

Section II [Descriptive Questions]: 1.5 Marks Each

1. Write a Node Snippet to display the current day and time on Node Engine With Your Name Concatenated With a Welcome Message. Write a Node Snippet to check two given numbers and return true if one of the number is 50 or if their sum is 50.

1) Current Day and Time with my Name and Welcome Message

```
const now = new Date();
const days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday'];
const currentDay = days[now.getDay()];
const currentTime = now.toLocaleTimeString();
const name = 'Srigowri N';

console.log(`Hello ${name}! Welcome to Node.js. Today is ${currentDay}, and the
current time is ${currentTime}.`);
```

2) Snippet to check two given numbers and return true if one of the number is 50 or if their sum is 50.

```
function checkNumbers(num1, num2) {
    if (num1 === 50 || num2 === 50 || (num1 + num2) === 50) {
        return true;
    } else {
        return false;
    }
}

console.log(checkNumbers(50, 10));  // true
console.log(checkNumbers(20, 25));  // false
```

2. Write a Node Snippet to Illustrate Process Global Object to retrieve

a. How much Memory is getting used

b. Get the Process ID

c. How much Time it took to get the Process in Action

d. How to Kill a Process and Abort it.

```
// a. How much Memory is getting used (in bytes)
const memoryUsage = process.memoryUsage();
console.log('Memory Usage:', memoryUsage);

// b. Get the Process ID
const pid = process.pid;
console.log('Process ID:', pid);

// c. How much Time it took to get the Process in Action (uptime in seconds)
const uptime = process.uptime();
console.log('Process Uptime (seconds):', uptime);

// d. How to Kill a Process and Abort it
// You can terminate the process by calling process.exit()
// Uncomment the next line to terminate the process immediately
// process.exit(0);  // 0 means successful exit

/*
If you want to kill another process by PID (from within Node),
you can use process.kill(pid), for example:
process.kill(pid); // kills current process, same as process.exit()
```

```
OR
process.kill(someOtherPid); // kills another process with id someOtherPid
*/

console.log('If you uncomment process.exit(), the process will terminate immediately.');
```

3. Write a Node Snippet to

a. Get the hostname of Given URL

b. Get the pathname given in the URL

c. Get the PortNo and protocol of the URL

d. Convert the Given List of URL's into JSON

```
const url = require('url');
// Sample URL to work with
const sampleUrl = 'https://example.com:8080/path/name?query=test';

// a. Get the hostname of given URL
const hostname = new URL(sampleUrl).hostname;
console.log('Hostname:', hostname);

// b. Get the pathname given in the URL
const pathname = new URL(sampleUrl).pathname;
console.log('Pathname:', pathname);

// c. Get the port number and protocol of the URL
const port = new URL(sampleUrl).port || 'default'; // if no port specified, show 'default'
const protocol = new URL(sampleUrl).protocol;
console.log('Port:', port);
```

```
console.log('Protocol:', protocol);

// d. Convert the given list of URLs into JSON
const urlList = [
  'https://google.com/search',
  'http://localhost:3000/api',
  'ftp://example.org/resource'
];

// Map URLs to objects containing hostname, pathname, port, protocol
const urlObjects = urlList.map(link => {
  const parsed = new URL(link);
  return {
    href: parsed.href,
    hostname: parsed.hostname,
    pathname: parsed.pathname,
    port: parsed.port || 'default',
    protocol: parsed.protocol
  };
});

// Convert to JSON string
const jsonResult = JSON.stringify(urlObjects, null, 2);
console.log('URL List in JSON:\n', jsonResult);
```

4. Write a Node Snippet to export and import

a. A function which shows the current system date and time [DateTime.js].

b. A function called as a lion using function expression, body containing roar in it [Animal.js].

c. Create a [App.js] using both DateTime and Animal By Importing its functionality.

```
// DateTime.js
function showDateTime() {
    const now = new Date();
    console.log('Current Date and Time:', now.toString());
}
module.exports = showDateTime;

// Animal.js
const lion = function() {
    console.log('Roar!');
};
module.exports = lion;

// App.js
const showDateTime = require('./DateTime');
const lion = require('./Animal');
showDateTime();  // Show current date and time
lion();          // Lion roar
```

Console:

node App.js

o/p: Current Date and Time: Tue Jul 15 2025 15:30:45 GMT+0530 (India Standard Time)
Roar!

5. Write a Node Snippet to make use of http module

a. To Create a Server which takes req and res

b. Body Containing a Welcome Message with Current System Date and Time.

c. Listening on a Port 3000

```
const http = require('http');

const server = http.createServer((req, res) => {
    // Get current date and time
    const now = new Date();

    // Set response header content type to plain text
    res.writeHead(200, { 'Content-Type': 'text/plain' });

    // Send response body with welcome message and current date/time
    res.end(`Welcome to Node.js! Current Date and Time: ${now.toString()}`);
});

// Server listens on port 3000
server.listen(3000, () => {
    console.log('Server running at http://localhost:3000/');
});
```

6. Write a Node Snippet on Event Emitters

a. Create a Class called as Dog extending EventEmitter.And Food Class.

b. Create an Instance of Dog using ON event (‘chew’) and passing Food reference to it.

c. Add ON Event (‘bark’) on Dog Class, If Triggered Get the Console.log stating (‘ Who’s at the Door ‘);

```
const EventEmitter = require('events');

// Food class
class Food {
    constructor(name) {
        this.name = name;
    }
}

// Dog class extends EventEmitter
class Dog extends EventEmitter {}

// Create instance of Dog
const dog = new Dog();

// Create instance of Food
const bone = new Food('Bone');

// Listener for 'chew' event, receiving food object
dog.on('chew', (food) => {
    console.log(`Dog is chewing on: ${food.name}`);
});
```

```
// Listener for 'bark' event
dog.on('bark', () => {
    console.log("Who's at the Door");
});

// Trigger the events
dog.emit('chew', bone);  // outputs: Dog is chewing on: Bone
dog.emit('bark');        // outputs: Who's at the Door
```

7. Write a Node Snippet [Functions] to make use of http module

a. Route to index.html [if (request.url == ‘/’)

b. Route to foo.html [if(request.url == ‘/foo’)

c. Modify the Same Using Switch Statements for Routes.

```
1) Using if for routing
const http = require('http');
const fs = require('fs');
const path = require('path');

function serveFile(res, filename) {
    const filePath = path.join(__dirname, filename);
    fs.readFile(filePath, (err, data) => {
        if (err) {
            res.writeHead(404);
            res.end('File not found');
        } else {
            res.writeHead(200, { 'Content-Type': 'text/html' });
            res.end(data);
```

```
        }
    });
}

const server = http.createServer((req, res) => {
    if (req.url === '/') {
        serveFile(res, 'index.html');
    } else if (req.url === '/foo') {
        serveFile(res, 'foo.html');
    } else {
        res.writeHead(404);
        res.end('Page not found');
    }
});

server.listen(3000, () => {
    console.log('Server running at http://localhost:3000/');
});
```

2) Switch statements for routing

```
const http = require('http');
const fs = require('fs');
const path = require('path');

function serveFile(res, filename) {
    const filePath = path.join(__dirname, filename);
    fs.readFile(filePath, (err, data) => {
        if (err) {
            res.writeHead(404);
            res.end('File not found');
        } else {
```

```
            res.writeHead(200, { 'Content-Type': 'text/html' });
            res.end(data);
        }
    });
}

const server = http.createServer((req, res) => {
    switch(req.url) {
        case '/':
            serveFile(res, 'index.html');
            break;
        case '/foo':
            serveFile(res, 'foo.html');
            break;
        default:
            res.writeHead(404);
            res.end('Page not found');
    }
});

server.listen(3000, () => {
    console.log('Server running at http://localhost:3000/');
});
```

8. Write a Node Snippet to

a. Insert Documents for Department Collection.

b. Find All Documents Available in Department Collection.

c. Update Department Document using ID

d. Remove Department Document

```
const { MongoClient, ObjectId } = require('mongodb');

const url = 'mongodb://localhost:27017';
const dbName = 'companyDB';

async function connectToDB() {
  const client = new MongoClient(url);
  await client.connect();
  const db = client.db(dbName);
  return { client, db };
}

// a. Insert Documents into Department Collection
async function insertDepartments(departmentCollection) {
  const insertResult = await departmentCollection.insertMany([
    { name: 'HR', location: 'New York' },
    { name: 'Engineering', location: 'San Francisco' },
    { name: 'Sales', location: 'Chicago' }
  ]);
  console.log('Inserted documents:', insertResult.insertedIds);
  return insertResult.insertedIds;
}
```

```
// b. Find All Documents in Department Collection
async function findAllDepartments(departmentCollection) {
  const allDepartments = await departmentCollection.find({}).toArray();
  console.log('All departments:', allDepartments);
  return allDepartments;
}

// c. Update Department Document by ID
async function updateDepartmentById(departmentCollection, id, updatedFields) {
  const updateResult = await departmentCollection.updateOne(
    { _id: new ObjectId(id) },
    { $set: updatedFields }
  );
  console.log(`Updated ${updateResult.modifiedCount} document(s)`);
}

// d. Remove Department Document
async function removeDepartment(departmentCollection, filter) {
  const deleteResult = await departmentCollection.deleteOne(filter);
  console.log(`Deleted ${deleteResult.deletedCount} document(s)`);
}

async function main() {
  const { client, db } = await connectToDB();
  const departmentCollection = db.collection('Department');

  try {
    // Insert
    const insertedIds = await insertDepartments(departmentCollection);
```

```
    // Find
    await findAllDepartments(departmentCollection);

    // Update - update first inserted document's location to 'Boston'
    const firstId = insertedIds['0'];
    await updateDepartmentById(departmentCollection, firstId, { location: 'Boston' });

    // Remove - delete document where name is 'Sales'
    await removeDepartment(departmentCollection, { name: 'Sales' });

  } catch (error) {
    console.error('Error:', error);
  } finally {
    await client.close();
    console.log('Connection closed');
  }
}

main();
```

9. Explain REPL With Sample Illustrations.

Node.js REPL stands for Read-Eval-Print Loop. It is a simple program that lets you type JavaScript code directly into your computer’s terminal or command prompt and instantly see the results.

Read: Reads the user input.
Eval: Evaluates the input as JavaScript.
Print: Outputs the result of the evaluation.
Loop: Repeats this process until the user exits.

Why is REPL useful?

You can quickly try out small pieces of JavaScript code without creating a file.

It’s useful when you want to test ideas or understand how something works.

Great for learning JavaScript and Node.js in a hands-on way.

Helps with debugging by letting you run tests immediately.

To Start REPL

Terminal: node

Node.js REPL prompt:

```
> const a = 15;
undefined
> const b = 25;
undefined
> a * b
375
> function square(num) {
...   return num * num;
... }
undefined
> square(8)
64
> let message = "Node.js REPL is fun!";
undefined
> message.toUpperCase()
'NODE.JS REPL IS FUN!'
> const arr = [2, 4, 6, 8, 10];
undefined
> arr.map(x => x * 3)
[ 6, 12, 18, 24, 30 ]
> exit
```