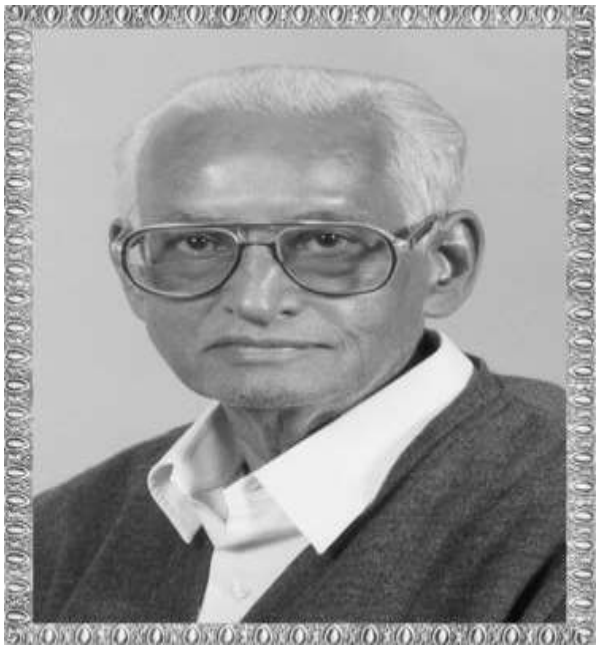
SHREE H.N.SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University and G.T.U)

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645



Lt. Shree Chimanbhai Shukla

MScIT SEM-3 :: CS-14: Node.js

Shree H.N.Shukla College
Street No. 2, Vaishali Nagar,
Nr. Amrapali Under Bridge,
Raiya Road, Rajkot.
Ph. (0281)2440478, 2472590

Shree H.N.Shukla College
Street No. 3, Vaishali Nagar,
Nr. Amrapali Under Bridge,
Raiya Road, Rajkot.
Ph. (0281)2471645

Shree H.N.Shukla College of I.T & Management "Sky is the Limit"

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

CS – 14: NODE JS

MSc(IT) SEMESTER : 03 :: CS-14: Node.js		
No	**Topics**	**Details**
4.	**Event, Database Connectivity**	• EventEmitter class • Methods and Events of EvenEmitter Class • Returning event emitter • Extend EventEmitter Class • Passing Arguments and 'this' to listeners • Asynchronous and Synchronous call • Handle Events only Once, Error Events • Connection string for database connectivity, • Configuring, Working with insert, select command, Updating records, Deleting records, Drop tables, Ordered Result Set

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

EventEmitter Class in Node.js

What is the EventEmitter class in Node.js?

The EventEmitter class is a core part of the Node.js event-driven architecture. It allows objects to emit events and handle those events using listener functions.

Example:

```
const EventEmitter = require('events');
class MyEmitter extends EventEmitter {}

const myEmitter = new MyEmitter();
myEmitter.on('event', () => {
  console.log('An event occurred!');
});
myEmitter.emit('event');
```

Output:

```
An event occurred!
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

What are the methods and events of the EventEmitter class?

Method/Event	Description
`on(event, listener)`	Adds a listener for the specified event.
`emit(event, [...args])`	Emits the specified event with arguments.
`once(event, listener)`	Adds a one-time listener for the event.
`removeListener(event, listener)`	Removes the specified listener from the event.
`removeAllListeners([event])`	Removes all listeners for the specified event.
`listeners(event)`	Returns an array of listeners for the specified event.
`listenerCount(emitter, event)`	Returns the number of listeners for the specified event.

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

function responseToEvent() {
  console.log('Event received!');
}

myEmitter.on('event', responseToEvent);
console.log(myEmitter.listenerCount('event')); // Output: 1
myEmitter.emit('event'); // Output: Event received!
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Node.js allows us to create and handle custom events easily by using events module. Event module includes EventEmitter class which can be used to raise and handle custom events.

```
// get the reference of EventEmitter class of events module
var events = require('events');

//create an object of EventEmitter class by using above reference
var em = new events.EventEmitter();

//Subscribe for FirstEvent
em.on('FirstEvent', function (data) {
   console.log('First subscriber: ' + data);
});

// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');
```

- First import the 'events' module and then create an object of EventEmitter class.
- Specify event handler function using on() function.
- The on() method requires name of the event to handle and callback function which is called when an event is raised.
- The emit() function raises the specified event.
- First parameter is name of the event as a string and then arguments.
- An event can be emitted with zero or more arguments.
- We can specify any name for a custom event in the emit() function.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Use of addListener() methods to subscribe for an event.

```
var emitter = require('events').EventEmitter;

var em = new emitter();

//Subscribe FirstEvent
em.addListener('FirstEvent', function (data) {
   console.log('First subscriber: ' + data);
});

//Subscribe SecondEvent
em.on('SecondEvent', function (data) {
   console.log('First subscriber: ' + data);
});

// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');

// Raising SecondEvent
em.emit('SecondEvent', 'This is my second Node.js event emitter example.');
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

The following table lists all the important methods of EventEmitter class.

EventEmitter Methods	Description
emitter.addListener(event, listener)	Adds a listener to the end of the listeners array for the specified event. No checks are made to see if the listener has already been added.
emitter.on(event, listener)	Adds a listener to the end of the listeners array for the specified event. No checks are made to see if the listener has already been added. It can also be called as an alias of emitter.addListener()
emitter.once(event, listener)	Adds a one time listener for the event. This listener is invoked only the next time the event is fired, after which it is removed.
emitter.removeListener(event, listener)	Removes a listener from the listener array for the specified event. Caution: changes array indices in the listener array behind the listener.
emitter.removeAllListeners([event])	Removes all listeners, or those of the specified event.
emitter.setMaxListeners(n)	By default EventEmitters will print a warning if more than 10 listeners are added for a particular event.
emitter.getMaxListeners()	Returns the current maximum

	listener value for the emitter which is either set by emitter.setMaxListeners(n) or defaults to EventEmitter.defaultMaxListeners.
emitter.listeners(event)	Returns a copy of the array of listeners for the specified event.
emitter.emit(event[, arg1][, arg2][, ...])	Raise the specified events with the supplied arguments.
emitter.listenerCount(type)	Returns the number of listeners listening to the type of event.

Common Patterns for EventEmitters

There are two common patterns that can be used to raise and bind an event using EventEmitter class in Node.js.

1. Return EventEmitter from a function
2. Extend the EventEmitter class

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

How do you return an event emitter?

To return an event emitter from a function, create an instance of EventEmitter and return it. This allows the caller to register event listeners.

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

function getEmitter() {
  const emitter = new EventEmitter();
  process.nextTick(() => {
    emitter.emit('start');
  });
  return emitter;
}

const myEmitter = getEmitter();
myEmitter.on('start', () => {
  console.log('Started!');
});
```

Output:

```
Started!
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

How do you extend the EventEmitter class?

You can extend the EventEmitter class using class inheritance.

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

class MyEmitter extends EventEmitter {}

const myEmitter = new MyEmitter();
myEmitter.on('event', () => {
  console.log('Event triggered!');
});
myEmitter.emit('event');
```

Output:

```
Event triggered!
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

How do you pass arguments and 'this' to listeners?

You can pass arguments to listeners by including them in the emit method call. The this context is automatically bound to the instance of EventEmitter.

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

class MyEmitter extends EventEmitter {
  triggerEvent() {
    this.emit('event', 'argument1', 'argument2');
  }
}

const myEmitter = new MyEmitter();
myEmitter.on('event', function(arg1, arg2) {
  console.log(`Event received with args: ${arg1}, ${arg2}`);
});
myEmitter.triggerEvent();
```

Output:

```
Event received with args: argument1, argument2
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Passing Arguments to Listeners

We can pass arguments to the event listeners when emitting events:

```
eventEmitter.on('start', (number) => {
   console.log(`Event started with number: ${number}`);
});

eventEmitter.emit('start', 42);
```

Output:

```
Event started with number: 42.
```

Multiple Listeners

We can attach multiple listeners to a single event:

```
eventEmitter.on('data', (data) => {
   console.log(`Received data: ${data}`);
});

eventEmitter.on('data', (data) => {
   console.log(`Logging data: ${data}`);
});

eventEmitter.emit('data', 'Hello, World!');
```

Output:

```
Received data: Hello, World!
Logging data: Hello, World!
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

What is the difference between asynchronous and synchronous calls?

Asynchronous and synchronous calls are fundamental concepts in Node.js that dictate how functions execute, particularly in relation to I/O operations.

Understanding these concepts is crucial for building efficient applications, especially when managing tasks like library book management.

Synchronous Calls

- Synchronous calls in Node.js block the execution of code until the operation completes.
- This means that the program waits for the current operation to finish before moving on to the next line of code.
- While this can simplify code flow, it can lead to performance issues, especially in I/O-bound tasks, as it prevents the application from handling other requests during this wait time.

Example: Synchronous Book Management

Imagine a simple function that retrieves a book's details from a database synchronously:

```
// book-issue.js
const fs = require('fs');

function getBookDetailsSync(bookId) {
   // Synchronously read book details from a file
   const data = fs.readFileSync(`books/${bookId}.json`, 'utf8');
   return JSON.parse(data);
}

const book = getBookDetailsSync('1');
console.log(book);
```

In this example, the application will pause at fs.**readFileSync** until the book details are fully read from the file.
If this operation takes time (e.g., due to a large file), the application becomes unresponsive during that period.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Asynchronous Calls

- In contrast, asynchronous calls allow the program to continue executing other code while waiting for an operation to complete.
- This is achieved through callbacks, promises, or **async/await** syntax.
- Asynchronous operations do not block the event loop, enabling the application to handle multiple requests concurrently, which is particularly beneficial for I/O-bound tasks.

Example: Asynchronous Book Management

```
// book-mgmt.js
const fs = require('fs').promises;

async function getBookDetailsAsync(bookId) {
  try {
    // Asynchronously read book details from a file
    const data = await fs.readFile(`books/${bookId}.json`, 'utf8');
    return JSON.parse(data);
  } catch (error) {
    console.error('Error reading book details:', error);
  }
}

getBookDetailsAsync('1').then(book => {
  console.log(book);
});
```

- In this asynchronous example, the getBookDetailsAsync function does not block the execution.
- While waiting for the file read operation to complete, the application can continue to handle other tasks.
- The use of async/await provides a clean syntax that resembles synchronous code while maintaining the benefits of asynchronous execution.

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

How do you handle events only once, including error events?

- **Once Handling:** Use the once method to handle an event only once.
- **Error Events:** Emit an error event and listen for it using on('error', listener).

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

myEmitter.once('event', () => {
  console.log('This will be called once.');
});
myEmitter.emit('event');
myEmitter.emit('event'); // Won't trigger the listener again

myEmitter.on('error', (err) => {
  console.error('Error occurred:', err);
});
myEmitter.emit('error', new Error('Something went wrong'));
```

Output:

```
This will be called once.
Error occurred: Error: Something went wrong
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

Database Connectivity in Node.js

How do you configure a connection string for database connectivity in Node.js?

A connection string is a string that specifies information about a data source and the means of connecting to it.

Example (MongoDB):

```
const mongoose = require('mongoose');
const connectionString = 'mongodb://username:password@localhost:27017/mydatabase';

mongoose.connect(connectionString, { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => console.log('Database connected!'))
  .catch(err => console.error('Database connection error:', err));
```

Output:

```
Database connected!
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

How do you work with the insert command in Node.js?

To insert records into a database, you can use the `insertOne` or `insertMany` methods in MongoDB.

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

const userSchema = new mongoose.Schema({
  name: String,
  age: Number,
});

const User = mongoose.model('User', userSchema);

const newUser = new User({ name: 'John', age: 30 });
newUser.save()
  .then(() => console.log('User inserted!'))
  .catch(err => console.error('Insert error:', err));
```

Output:

```
User inserted!
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

How do you use the select command in Node.js?

To select records from a database, use the find method in MongoDB.

```
const User = mongoose.model('User', userSchema);

User.find({ age: { $gt: 20 } })
  .then(users => console.log('Users found:', users))
  .catch(err => console.error('Select error:', err));
```

Output:

```
Users found: [{ _id: '...', name: 'John', age: 30 }]
```

How do you update records in Node.js?

To update records, use the updateOne or updateMany methods in MongoDB.

```
User.updateOne({ name: 'John' }, { $set: { age: 31 } })
  .then(() => console.log('User updated!'))
  .catch(err => console.error('Update error:', err));
```

Output:

```
User updated!
```

2 – Vaishalinagar,
Nr.Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2440478

3 – Vaishalinagar,
Nr. Amrapali Railway Under Bridge
Raiya Road,
Rajkot – 360001.
Ph No. 0281- 2471645

How do you delete records in Node.js?

To delete records, use the ***deleteOne*** or ***deleteMany*** methods in MongoDB.

```
User.deleteOne({ name: 'John' })
  .then(() => console.log('User deleted!'))
  .catch(err => console.error('Delete error:', err));
```

***Output*:**

```
User deleted!
```

How do you drop tables in Node.js?

To drop a collection (equivalent to dropping a table), use the drop method in MongoDB.

```
mongoose.connection.dropCollection('users')
  .then(() => console.log('Collection dropped!'))
  .catch(err => console.error('Drop error:', err));
```

Output:

```
Collection dropped!
```

How do you get an ordered result set in Node.js?

To get an ordered result set, use the sort method in MongoDB.

```
User.find().sort({ age: 1 })
  .then(users => console.log('Users sorted by age:', users))
  .catch(err => console.error('Sort error:', err));
```

Output:

Users sorted by age: [{ _id: '...', name: 'Jane', age: 25 },
{ _id: '...', name: 'John', age: 30 }]