NodeJS Slip Programs

Unit 2

1. Write Node.js application to demonstrate following properties/methods of buffer module – concat, compare, copy, equals, fill, includes, indexOf, length, slice, write.

```
=>// Importing the 'Buffer' module

const { Buffer } = require('buffer');


// Create a buffer with "Hello"

const buffer1 = Buffer.from('Hello');


// Demonstrate properties/methods

console.log('Concatenated Buffer:', Buffer.concat([buffer1, Buffer.from(' World')]).toString());

console.log('Comparison Result:', buffer1.compare(Buffer.from('Hello')));

console.log('Copied Buffer:', Buffer.from(buffer1).toString());

console.log('Are buffers equal?', buffer1.equals(Buffer.from('Hello')));

console.log('Filled Buffer:', Buffer.alloc(5).fill('A').toString());

console.log('Does buffer1 include "lo"?', buffer1.includes('lo'));

console.log('Index of "e" in buffer1:', buffer1.indexOf('e'));

console.log('Length of buffer1:', buffer1.length);

console.log('Sliced Buffer:', buffer1.slice(1, 3).toString());


// Create a writable buffer

const writableBuffer = Buffer.alloc(8);

writableBuffer.write('Node.js', 'utf-8');
```

```
console.log('Written Buffer:', writableBuffer.toString());
```

2. Write Node.js application(s)

A. that uses user defined module that contains a function to mask the 10-digit credit card number except last 3 digits.

=>**// creditCardModule.js**

```
function maskCreditCard(cardNumber) {
   if (typeof cardNumber !== 'string' || cardNumber.length !== 16) {
      throw new Error('Invalid credit card number format');
   }
// Masking all digits except the last 3
   const maskedNumber = '************' + cardNumber.slice(-3);
   return maskedNumber;
}
module.exports = {
   maskCreditCard,
};
```

// appCreditCard.js

```
const creditCardModule = require('./creditCardModule');

const creditCardNumber = process.argv[2];

if (creditCardNumber && creditCardNumber.length === 16 &&
!isNaN(creditCardNumber)) {
```

```
  const maskedCardNumber =
creditCardModule.maskCreditCard(creditCardNumber);
  console.log('Masked:', maskedCardNumber);
} else {
  console.log('Please provide a valid 16-digit credit card number as a
command line argument.');
}
```

****

B. that uses user defined module that contains a function that displays whether the entered year is a leap year or not.

=>**//leapyearModule.js**

```
function isLeapYear(year) {
  if (typeof year !== 'number' || isNaN(year)) {
    throw new Error('Invalid input. Please enter a valid year.');
  }

  if ((year % 4 === 0 && year % 100 !== 0) || (year % 400 === 0)) {
    return true;
  } else {
    return false;
  }
}

module.exports = {
  isLeapYear,
};
```

// appLeapYear.js

```
const leapYearModule = require('./leapYearModule');

const year = parseInt(process.argv[2]);

console.log(`${year} is ${leapYearModule.isLeapYear(year) ? 'a Leap Year' : 'not a Leap Year'}`);
```

****

3. Write Node.js application(s)

A. that uses user defined module to accept a string and return the count of vowels in that string.

=>**//vowelCountModule.js**

```
function countVowels(inputString) {
   if (typeof inputString !== 'string') throw new Error('Invalid input. Please provide a valid string.');
   return inputString.split('').filter(char => 'aeiouAEIOU'.includes(char)).length;
}

module.exports = {
   countVowels,
};
```

// appVowelCount.js

```
const vowelCountModule = require('./vowelCountModule');

const inputString = process.argv[2];


if (inputString) {
   console.log(`Number of vowels in "${inputString}":`, vowelCountModule.countVowels(inputString));
} else {
   console.log('Please provide a string as a command line argument.');
```

```
}
```

B. that uses user defined module to check if the entered string or a number is palindrome or not.

=> **// palindromeModule.js**

```
function isPalindrome(input) {
   const stringToCheck = String(input).toLowerCase().replace(/[^a-zA-Z0-9]/g, '');
   const reversedString = stringToCheck.split('').reverse().join('');
   return stringToCheck === reversedString;
}
module.exports = {
   isPalindrome,
};
```

// appPalindrome.js

```
const palindromeModule = require('./palindromeModule');
const input = process.argv[2];
const result = palindromeModule.isPalindrome(input);
console.log(`"${input}" is ${result ? 'a Palindrome' : 'not a Palindrome'}`);
```

****

4. Write Node.js application(s)

A. that uses user defined module to return the factorial of a given number.

=>**// factorialModule.js**

```
function calculateFactorial(number) {
   if (number === 0 || number === 1) {
      return 1;
   }
```

```
 return number * calculateFactorial(number - 1);
}
module.exports = {
   calculateFactorial,
};
```

//factorial.js

```
const factorialModule = require('./ factorialModule.js');
const number = parseInt(process.argv[2]);

if (!isNaN(number) && number >= 0) {
   console.log(`Factorial of ${number}:`,
factorialModule.calculateFactorial(number));
} else {
   console.log('Please provide cmd argument.');
}
```

B. that uses user defined module circle.js which exports functions area () and circumference () and displays calculated values on the console. Accept radius from the user

=>**//circle.js**

```
function area(radius) {
   return Math.PI * radius * radius;
}
function circumference(radius) {
   return 2 * Math.PI * radius;
}
module.exports = {
   area,
```

```
    circumference,
};

// appCircle.js
const circleModule = require('./circle');

const radius = parseFloat(process.argv[2]);

if (!isNaN(radius) && radius >= 0) {
    console.log('Area:', circleModule.area(radius));
    console.log('Circumference:', circleModule.circumference(radius));
} else {
    console.log('Please provide a non-negative radius as a command line argument.');
}
```

****

5. Write Node.js application(s)

A. that uses user defined module to find area of a rectangle and display details on console.

=>**//reactangleModule.js**

```
function calculateRectArea(length, width) {
    return (length && width && length > 0 && width > 0) ? length * width : 0;
}
module.exports = {
    calculateRectArea,
};
```

//rectangle.js

```
var rectangleModule = require('./5.a.rectModule.js');


var length = parseFloat(process.argv[2]);
var width = parseFloat(process.argv[3]);


var area = rectangleModule.calculateRectArea(length, width);
console.log(area ? `Rectangle Details:\n  Length: ${length}\n  Width: ${width}\n  Area: ${area}` : 'Invalid input.');
```

B. that uses user defined module ‘calculator’ which has functions – add, subtract, multiply, divide. Accept 2 numbers and an operation from the user.

=>**//calculator.js**

```
function add(a,b){
   return a+b;
}
function sub(a,b){
return a-b;
}
function mult(a,b){
   return a*b;
}
function div(a,b){
   return a/b;
}
exports = add;
exports = sub;
```

```
exports = mult;
exports = div;
```

//appCalculator.js

```
const cal = require('./calculator.js');
var a = parseFloat(process.argv[2]);
var b = parseFloat(process.argv[3]);

var operator = process.argv[4];
switch(operator){
   case '+':
     console.log('Addition is: '+cal.add(a,b));
   break;
   case '-':
      console.log('Subtraction is: '+cal.sub(a,b));
   break;
   case '*':
      console.log('Multiplication is: '+cal.mult(a,b));
   break;
   case '/':
      console.log('Division is: '+cal.div(a,b));
   break;
}
```

****

Unit 3

6. Write Node.js application from the scratch using NPM. Demonstrate the use of third-party dependencies using all NPM commands – add dependency to package.json, install package, update package, uninstall package.

=>const _ = require('lodash');

```
const numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5];
const sortedNumbers = _.sortBy(numbers);

console.log('Original Numbers:', numbers);
console.log('Sorted Numbers:', sortedNumbers);
```

****

Unit 4

7. Create Node.js web server to demonstrate – handling HTTP requests, use of query strings and returning HTTP & JSON response.

```
=>var http = require('http');
var url = require('url');

var server = http.createServer(function (req, res) {

  //Query String
  res.writeHead(200, { 'Content-Type': 'text/html' });
  var string = url.parse(req.url, true).query;
  var name = string['name'];
  if (name) {
    res.end('Hi', + name);
  }
```

```
//HTTP Handling Response
if (req.url == '/java') {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<html><body><h2>JAVA</h2><p> This is java.Your subject for sem 4</p></body></html>');
    res.end();
}
else {
    res.writeHead(400, 'Invalid Request');
    res.end('<html><body><h2>Invalid Request!</h2></body></html>');
}
//HTTP and JSON Response
if (req.url == '/data') {
    res.writeHead(200, { 'Content-Type': 'application/json' });
    res.write(JSON.stringify({ message: "Hello World" }));
    res.end();
}
// json object (obj.arr)
var jsonData = {
    "subjects": [
        { "name": "Node.js", "marks": "22" },
        { "name": "Java", "marks": "20" },
        { "name": "PHP", "marks": "21" }
    ]
};
res.writeHead(200, { 'Content-Type': 'application/json' });
for (i = 0; i < jsonData.subjects.length; i++) {
```

```
        res.write(jsonData.subjects[i].name);
        res.write('\t');
        res.write(jsonData.subjects[i].marks);
        res.write('\n');
    }
});
server.listen(7080);
console.log('Node.js web server is running on port 7080...');
```

****

8. Create a simple web server using Node.js that displays the college information on a web page.

=>**//collegeInfo.html**

```
<html>
<head>
<title>BMCC</title>
</head>
<body>
<h1>Brihan Maharashtra College of Commerce (BMCC) - Pune</h1>
    <p>Location: Pune, Maharashtra, India</p>
    <p>Established: 1943</p>
    <p>Website: <a href="https://www.bmcc.ac.in/"
target="_blank">https://www.bmcc.ac.in/</a></p>
</body>
</html>
```

//collegeInfo.js

```
var http = require('http');
var fs = require('fs');
```

```
http.createServer(function (req,res){
   fs.readFile('2.clg.html', function(err, data){
      res.writeHead(200,{'Content-Type': 'text/html'});
      res.write(data);
      return res.end();
   });
}).listen(8080);
```

****

9. Write Node.js application that accepts first name, last name, and date of birth (in dd/mm/yyyy format) from the user. Concatenate first name & last name and calculate the age. Greet the user with a full name along with the age, on a web page. Note: Use readline to accept input from the user.

```
=>var readline = require('readline');
var http = require('http');


var rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});


rl.question('Enter first name: ', function(firstName) {
  rl.question('Enter last name: ', function(lastName) {
    rl.question('Enter date of birth (yyyy): ', function(dob){
      rl.close();


      var fullName = `${firstName} ${lastName}`;
      var currentDate = new Date();
```

```
    var age = currentDate.getFullYear() - dob;

    var server = http.createServer(function(req, res){
      res.writeHead(200, { 'Content-Type': 'text/html' });
      res.end(`<h1>Hello ${fullName}!</h1><p>Your age is ${age} years.</p>`);
    });
    server.listen(8080);
    console.log("HTTP Request is running....");

  });
 });
});
```

****

Unit 5

10. Write Node.js application(s)

A. To demonstrate streams and pipes.

```
=>const fs = require("fs");
fs.writeFileSync('output.txt', 'This data will be written to a file using write stream.', 'UTF8');
const readStream = fs.createReadStream('output.txt');
const writeStream = fs.createWriteStream('output-pipe.txt');
readStream.pipe(writeStream);
fs.createReadStream('input-pipe.txt').pipe(writeStream);
writeStream.on('finish', () => console.log("Piping completed."));
console.log("Program Completed Successfully");
```

B. To demonstrate all the methods of readline module.

```
=>
const readline = require('readline');

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

// Method 1: Question method
rl.question('What is your name? ', (name) => {
  console.log(`Hello, ${name}!`);~

  // Method 2: Set prompt and use 'prompt' method
  rl.setPrompt('Enter a number: ');
  rl.prompt();

  // Method 3: Handle 'line' event
  rl.on('line', (input) => {
    console.log(`Received: ${input}`);

    // Method 4: Pause and resume the input stream
    rl.pause();
    setTimeout(() => {
      rl.resume();
      console.log('Resumed reading input.');
    }, 1000);
  });
```

```
  // Method 5: Write to the output stream
  rl.write('This is a message to the output stream.\n');

  // Method 6: Handle 'close' event
  rl.on('close', () => {
    console.log('Readline interface closed.');
  });

  // Method 7: Handle 'SIGINT' (Ctrl+C) event
  rl.on('SIGINT', () => {
    rl.question('Do you really want to exit? (yes/no) ', (answer) => {
      if (answer.toLowerCase() === 'yes') {
        rl.close();
      } else {
        rl.prompt();
      }
    });
  });
});
```

****

11. Write Node.js application(s)

A. To count the number of words in a file and display on the console.

```
=>var fs = require('fs');

fs.readFile('readFile.txt', 'utf-8', function (err, data) {
```

```
   if (err)
      throw err;
   cnt = 1;
   for (i = 0; i < data.length; i++) {
      if (data[i] == " " || data[i] == "\n") {
         cnt++;
      }
   }

   console.log("number of words are :", cnt)
});
```

B. To search a particular word in file and display the count on the console. Accept the word from the user, as a command line argument.

```
=>var fs = require('fs');
var word = process.argv[2];

fs.readFile('readFile.txt', 'utf-8', function (err, data) {
  if (err) {
    throw err;
  }

  var arr = data.split(word);
  console.log(`the word'${word}' is present ${arr.length-1}times`);
});
```

****

12. Write Node.js application(s)

A. To count the number of lines in a file and display the same on the console.

```
=>var fs = require('fs');

fs.readFile('readFile.txt', 'utf-8', function (err,data) {
   if (err)
      throw err;
   cnt=1;
   for(i=0;i<data.length;i++){
      if(data[i] == "\n") {
         cnt++;
      }
   }
   console.log("number of Lines are :" , cnt)

});
```

B. To count the number of vowels in a file and display the same on the console.

```
=>var fs = require('fs');

fs.readFile('readFile.txt', 'utf-8', function (err,data) {
   if (err)
      throw err;
   cnt=0;
   for(i=0;i<data.length;i++){
      if(data[i] == "a" || data[i] == "e" || data[i] == "i"|| data[i] == "o" || data[i]
== "u") {
```

```
        cnt++;
      }
   }
   console.log("number of vowel are :" , cnt)
});
```

****

13. Write separate Node.js application(s)

A. To check if a folder exists, create a new folder, display contents of an existing folder, rename a folder and remove a folder.

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

// Check if a folder exists
const checkFolderExists = (folderPath) => {
  return fs.existsSync(folderPath);
};

// Create a new folder
const createFolder = (folderPath) => {
  fs.mkdirSync(folderPath);
  console.log(Folder created at ${folderPath});
};

// Display contents of an existing folder
const displayFolderContents = (folderPath) => {
  const contents = fs.readdirSync(folderPath);
```

```
  console.log(Contents of ${folderPath}:);
  contents.forEach((item) => {
    console.log(item);
  });
};

// Rename a folder
const renameFolder = (oldPath, newPath) => {
  fs.renameSync(oldPath, newPath);
  console.log(Folder renamed from ${oldPath} to ${newPath});
};

// Remove a folder
const removeFolder = (folderPath) => {
  fs.rmdirSync(folderPath, { recursive: true });
  console.log(Folder removed at ${folderPath});
};

// Example usage
const folderPath = './exampleFolder';
if (!checkFolderExists(folderPath)) {
  createFolder(folderPath);
}
displayFolderContents(folderPath);
const newFolderPath = './renamedFolder';
renameFolder(folderPath, newFolderPath);
```

removeFolder(newFolderPath);

B. To check whether the given name is a directory or a file. If it is a file, truncate the content after 10 bytes. (Accept the name from the user)

=>const fs = require('fs');

const readline = require('readline');

```
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

// Check if the given path is a directory or a file
const checkPathType = (path) => {
  return fs.statSync(path).isDirectory() ? 'directory' : 'file';
};

// Truncate content of a file after 10 bytes
const truncateFileContent = (filePath) => {
  fs.readFile(filePath, 'utf8', (err, data) => {
    if (err) {
      console.error(err);
    } else {
      const truncatedContent = data.slice(0, 10);
      fs.writeFileSync(filePath, truncatedContent);
      console.log(Content truncated in ${filePath});
    }
  });
```

```
};


// Example usage
rl.question('Enter the path (file or directory): ', (path) => {
  const pathType = checkPathType(path);


  if (pathType === 'directory') {
    console.log(${path} is a directory.);
  } else {
    console.log(${path} is a file.);
    truncateFileContent(path);
  }
  rl.close();
});
```

****

14. Write Node.js application(s)

A. To accept two file names and the contents from the user and write to two files. Then append contents of a first file to a second file. Display the contents of a second file on the console.

i. Using streams

```
=>var fs = require('fs');
var readline = require('readline');


let rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});
```

```
// Accept file names and contents from the user
rl.question('Enter the first file name: ', (firstFileName) => {
  rl.question('Enter the content for the first file: ', (content1) => {
    rl.question('Enter the second file name: ', (secondFileName) => {
      rl.question('Enter the content for the second file: ', (content2) => {
        // Write to the first file using streams
        const stream1 = fs.createWriteStream(firstFileName);
        stream1.write(content1);
        stream1.end();

        // Append contents of the first file to the second file using streams
        const stream2 = fs.createWriteStream(secondFileName, { flags: 'a' });
        const readStream1 = fs.createReadStream(firstFileName);
        readStream1.pipe(stream2);

        // Display contents of the second file on the console using streams
        const readStream2 = fs.createReadStream(secondFileName);
        readStream2.on('data', (chunk) => {
          console.log(chunk.toString());
        });

        rl.close();
      });
    });
  });
```

```
});
```

ii. Using synchronous operations

=>// accept file names and contents from the user synchronously

```
const fs = require('fs');
const readlineSync = require('readline-sync');

// Accept file names and contents from the user synchronously
const firstFileName = readlineSync.question('Enter the first file name: ');
const content1 = readlineSync.question('Enter the content for the first file: ');
const secondFileName = readlineSync.question('Enter the second file name: ');
const content2 = readlineSync.question('Enter the content for the second file:
');

// Write to the first file synchronously
fs.writeFileSync(firstFileName, content1);

// Append contents of the first file to the second file synchronously
fs.appendFileSync(secondFileName, fs.readFileSync(firstFileName));

// Display contents of the second file on the console synchronously
console.log(fs.readFileSync(secondFileName).toString());

// accept file name and contents from the user async
const fs = require('fs');
const readline = require('readline');
```

```
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});
```

iii. Using asynchronous operations

```
=>// Accept file names and contents from the user asynchronously
rl.question('Enter the first file name: ', (firstFileName) => {
  rl.question('Enter the content for the first file: ', (content1) => {
    rl.question('Enter the second file name: ', (secondFileName) => {
      rl.question('Enter the content for the second file: ', (content2) => {
        // Write to the first file asynchronously
        fs.writeFile(firstFileName, content1, (err) => {
          if (err) throw err;

          // Append contents of the first file to the second file asynchronously
          fs.appendFile(secondFileName, content1, (err) => {
            if (err) throw err;

            // Display contents of the second file on the console asynchronously
            fs.readFile(secondFileName, (err, data) => {
              if (err) throw err;
              console.log(data.toString());
              rl.close();
            });
```

```
        });
      });
    });
  });
});
```

****

15. Write Node.js application(s) A. To read contents from two files and merge the contents into a third file by converting it into uppercase. Note: Use synchronous and asynchronous operations

=>

```
const fs = require('fs');
const readline = require('readline');

const rl = readline.createInterface({ input: process.stdin, output: process.stdout });

rl.question('Enter first file name, second file name, and output file name (e.g., file1.txt file2.txt output.txt): ', input => {
  rl.close();
  const [file1, file2, outputFile] = input.split(' ');

  const content1 = fs.readFileSync(file1, 'utf8');
  const content2 = fs.readFileSync(file2, 'utf8');
  const mergedContent = (content1 + '\n' + content2).toUpperCase();

  fs.writeFileSync(outputFile, mergedContent, 'utf8');
  console.log(`Contents merged and converted to uppercase. Written to ${outputFile}`);
```

```
});
```

****

16. Write separate Node.js application(s)

A. To search a particular word in a file and replace all the occurrences of that word with another word. Accept both the words from the user.

```
=> const fs = require('fs');
const readline = require('readline');


const rl = readline.createInterface({ input: process.stdin, output: process.stdout
});


rl.question('Enter file name, word to search, and word to replace (e.g., file.txt
oldWord newWord): ', (input) => {
  rl.close();
  const [fileName, searchWord, replaceWord] = input.split(' ');


  fs.readFile(fileName, 'utf8', function (err, data) {
    if (err) throw err;


    const replacedContent = data.split(searchWord).join(replaceWord);


    fs.writeFile(fileName, replacedContent, 'utf8', function (err) {
      if (err) throw err;
      console.log(`Occurrences of "${searchWord}" replaced with
"${replaceWord}".`);
    });
  });
```

});

B. To swap contents of two files. Accept files names as command line arguments.

=>const fs = require('fs');

const [file1, file2] = process.argv.slice(2);

// Read contents of file1

var content1 = fs.readFileSync(file1, 'utf8');

// Read contents of file2

var content2 = fs.readFileSync(file2, 'utf8');

// Write contents of file2 to file1

fs.writeFileSync(file1, content2, 'utf8');

// Write original contents of file1 to file2

fs.writeFileSync(file2, content1, 'utf8');

console.log(`Contents swapped between ${file1} and ${file2}.`);

****

17. Write separate Node.js application(s)

A. To display contents of a file in the reverse order. Accept file name as command line argument.

=>const fs = require('fs');

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

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

rl.question('Enter the file name: ', function(fileName) {
  const reversedData = fs.readFileSync(fileName, 'utf8').split('').reverse().join('');
  console.log('Contents in reverse order (Console):\n', reversedData);
  rl.close();
});
```

B. To display alternate characters from a file. Accept file name as command line argument.

```
=>const fs = require('fs');
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

// Function to display alternate elements from an array
function displayAlternate(arr, step) {
    return arr.filter((element, index) => index % step === 0).join(' ');
}
```

```
// Ask user for the file name
rl.question('Enter the file name: ', function(fileName) {
    const data = fs.readFileSync(fileName, 'utf8');

    // Display alternate characters, words, and lines
    console.log('Alternate Characters:', displayAlternate(Array.from(data), 2));
    console.log('Alternate Words:', displayAlternate(data.split(/\s+/), 2));
    console.log('Alternate Lines:\n', displayAlternate(data.split('\n'), 2));

    rl.close();
});
```

****

18. Write Node.js application to display contents of a file in the reverse order on browser using Node.js web server.

=>

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

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

rl.question('Enter the file name: ', function (fileName) {
    const reversedData = fs.readFileSync(fileName,
'utf8').split('').reverse().join('');
```

```
    // Create a simple HTTP server to display the reversed contents
    http.createServer((req, res) => {
        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.end(`<html><body><pre>${reversedData}</pre></body></html>`);
    }).listen(3000);

    console.log('Web server running at http://localhost:3000/');
    rl.close();
});
```

****

Unit 6

19. Write Node.js application that has an EventEmitter which will emit an event that contains information about the application's uptime, every second.

```
=>//import events module
var events = require('events');

//create an eventEmitter object
var timerEventEmitter = new events.EventEmitter();

let currentTime = 0;

//this will trigger the update event which passing second
setInterval(function (){
    currentTime++;
    timerEventEmitter.emit('update',currentTime);
},1000);
```

```
timerEventEmitter.on('update' , function(time){
   console.log('Message received from publisher');
   console.log(`${time} second(s) passed since the program started`);
});
```

****

20. Write separate Node.js application(s)

A. To raise and bind an event by returning EventEmitter object from a function.

=>

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

function LoopProcessor(num) {
   var e = new EventEmitter();
   setTimeout(function(){
      for (var i =1; i <= num; i++) {
         e.emit('BeforeProcess', i);
         console.log('Processing number:' +i);
         e.emit('AfterProcess', i);
      }
   }, 10)
   return e;
}
var lp = LoopProcessor(3);

lp.on('BeforeProcess', function(data){
   console.log("About to start the process for " + data);
});
```

```
lp.on('AfterProcess', function(data){
   console.log("Completed processing " + data);
});
```

B. To raise and bind an event by extending the EventEmitter class.

=>

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

class CustomEventEmitter extends EventEmitter {
  // Custom function to raise the event
  raiseEvent(data) {
    this.emit('customEvent', data);
  }
}

// Usage example
const emitter = new CustomEventEmitter();

// Binding the event handler
emitter.on('customEvent', (data) => {
  console.log('Event received with data:', data);
});

// Raising the event
emitter.raiseEvent({ message: 'Hello, world!' });
```

****

21. Write separate Node.js application(s)

A. To bind 2 listeners to a single event.

=>// File: bindListeners.js

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

// Create an instance of EventEmitter
const myEmitter = new EventEmitter();

// Event handler function 1
const listener1 = () => {
  console.log('Listener 1 called');
};

// Event handler function 2
const listener2 = () => {
  console.log('Listener 2 called');
};

// Bind listeners to a custom event 'customEvent'
myEmitter.on('customEvent', listener1);
myEmitter.on('customEvent', listener2);

// Emit the event
myEmitter.emit('customEvent');
```

B. To bind custom event 'receive_data' with 'data_receive_handler' function

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

```
// Create an instance of EventEmitter
const myEmitter = new EventEmitter();

// Custom event 'receive_data' handler function
const dataReceiveHandler = (data) => {
  console.log(`Data received: ${data}`);
};

// Bind the handler to the custom event 'receive_data'
myEmitter.on('receive_data', dataReceiveHandler);

// Emit the custom event with some data
myEmitter.emit('receive_data', 'Hello, this is the data!');
```

****

22. Write separate Node.js application(s) to countdown from 10 seconds to 0. Update the user every

second. Notify user when last 2 seconds are remaining and display a message “Time up!” at the

end.

A. Return EventEmitter object from a function

=>

```
var events = require('events');

function countDown(countDownTime) {
   var eventEmitter = new events.EventEmitter();

   //this will trigger the update event each passing second
   const timer = setInterval(function () {
```

```
        eventEmitter.emit('update',countDownTime);
        countDownTime--;

        if(countDownTime==0){
            clearInterval(timer);
            eventEmitter.emit('end');
        }

        if (countDownTime == 2) {
            eventEmitter.emit('end-soon');
        }
    }, 1000);
    return eventEmitter;
}

const myCountDown = countDown(10);

myCountDown.on('update' , (t) => {
    console.log(`time remaining: ${t} second(s)`);
});

myCountDown.on('end', () => {
    console.log('Time up!!');
});

myCountDown.on('end-soon', () =>{
```

```
    console.log('CountDown will end in 2 seconds');
});
```

B. Extend the EventEmitter class

```
=>var events = require('events');

const { EventEmitter } = require('stream');

class countDown extends EventEmitter {
    constructor(countDownTime) {
        super();
        var myEmitter = this;

        //this will trigger the update event each passing second
        const timer = setInterval(function () {
            myEmitter.emit('update', countDownTime);
            countDownTime--;

            if (countDownTime == 0) {
                clearInterval(timer);
                myEmitter.emit('end');
            }

            if (countDownTime == 2) {
                myEmitter.emit('end-soon');
            }
```

```
        }, 1000);
        return myEmitter;
    }
}

const myCountDown =  new countDown(10);

myCountDown.on('update', (t) => {
    console.log(`time remaining: ${t} second(s)`);
});

myCountDown.on('end', () => {
    console.log('Time up!!');
});

myCountDown.on('end-soon', () => {
    console.log('CountDown will end in 2 seconds');
});
```
****