Fs module in nodejs

1. It is builtin module using which we can read or write data into file
2. It has synchronous as well as asynchronous functions, every synchronous function name ends with sync.
3. Every asynchronous function has last parameter as callback function.
4. In callback function usually the first parameter we get is error
5. After successful completion of asynchronous function, its output will be automatically passed to callback function

Fs module functions

readFile(fname,callback function) fs.readFile("mydata.txt",(err,data)=>{ ///code goes here })	It reads entire file and pass the data to callback function
readFileSync(fname) var data=fs.readFileSync("mydata.txt")	It is a blocking function, which reads the entire file and return data
read(filediscriptor,buffer,bufferposition,length,fileposition,callback function) read(fd,buff,0,10,null,(err,bytesread,buffer)=>{}) in callback function bytesread is number of bytes read actually from the file, and buffer will get refence of buff, in which the data is stored	It reads length number of characters from the file, it starts reading of the file from given fileposition, and starts writing data in the buffer, from given bufferposition
createReadStream(fname,encoding)	When we want to do data streaming from a file, then create a read stream, while reding data, it generates data event, and when it ends it generates end event, and if error occurs then it generates error event

createWriteStream(fname,encoding)	It is used to write data in the destination, For writing data, use write event function , for ending the stream use end function,
Source.pipe(destination)	Pipe function will take care of synchronising readstream and writestream events, It will transfer data from source to destination
exists(fname,callback function) fs.exists(“mydata.txt”,(flag)=>{})	It checks whether file exists or not
stat(fname,callback function)	It checks the status of the file, It gives file properties like filesize, last access time, last modified time
open(fname,callback function)	It opens the file and passes file handle(file descriptor) to callback function
close(filename,()=>{})	

Note: to handle event on any object use on function.
To read file line by line, we nee to use readline module

readline.createInterface({ input:fs.createReadStream(fname) })	The interface will allow you to read the data from file line by line, after reading each line, it will generate line event, and when the file is end, then it generates close event
const readline=require("readline") const fs=require("fs") var rl=readline.createInterface({	

```
        input:fs.createReadStream("test.txt")
        })
        //var uname="admin"
        //var password="admin"

        var cnt=0;
        //for each line line event will get
        genarated
        rl.on("line",function(str){
          cnt++;
          console.log(cnt+ ": "+str);

        })

        rl.on("close",function(){
           console.log("Total number of lines
        : "+cnt)
})
```

http module
This module helps us to create a server, and handle request we receive from the server

createServer(function(req,resp){
//code for generating request and response
}
	It is asynchronous function to create a server, all the requests will be handled by the callback function
	Call back function takes 2 parameters req (request), and resp (response)

listen	It is used to start server at specified port number

To store all dependencies, and its version, we use package.json

1. To create a basic package.json file use command in command prompt
   D:\nodes\.....nodedemos\expressjsdemos>npm init
2. This command will show you multiple questions, select all drfault answers, it will generate package.json file, in current folder
3. In package.json file we can store all dependencies, and its versions, so that the same version will be installed on clients machine
4. For that add “dependency” key in the file manually

```
        {
          "name": "expressdemos",
          "version": "1.0.0",
          "description": "",
          "license": "ISC",
          "author": "Kishori",
          "type": "commonjs",
          "main": "index.js",
          "scripts": {
            "test": "echo \"Error: no test specified\" && exit 1"
          },
          "dependencies":{
```

```
        "express":"",
        "mysql":"^2.17.1"
      }
}
```

5. Then in command prompt give command
   D:\nodes\.....nodedemos\expressjsdemos>npm install
6. It will install all dependencies locally, and will save in current folder, in node_modules folder

7. The another way of doing same thing is use following command in command prompt
   D:\nodes\.....nodedemos\expressjsdemos>npm install mysql express
   Then also it will install these packages locally, in node_modules folder and automatically add dependencies key ,in package.json
8. Since we download these packages locally it can be used only in current folder, but if you want to install only once and use it anywhere on your machine, then install it globally, install any package globally we use -g flag
   D:\nodes\.....nodedemos\expressjsdemos>npm install -g mysql express

ExpressJS

1. Create a folder by name expressjsdemos
2. Open command prompt, and change folder to expressdemos folder
   D: \nodejsdemos\expressjsdemos>
3. Create package.json file, using npm init command
   D: \nodejsdemos\expressjsdemos> npm init
4. And then install epress and bosy-parser module by using npm install command, and see the change in package.jspn, dependencies key will be added in package.json file
   D: \nodejsdemos\expressjsdemos>npm install express body-parser

   To parse the data that we receive through request object using get/post method,
   We use body-parser middleware

   If the method is get, then data will be parsed and will be stored in req.query object
   If the method is post then the data will be stored in req.body object

   const bodyParser=require("body-parser")
   bodyParser.urlencoded({ extended: false }) if the data is in urlencoded form
   but if the data is in json format, then use
   bodyParser.json()

extended: false

1. Uses the Node.js built-in querystring library.
2. Only supports simple key-value pairs, NOT nested objects.
   Example: Input from form:
   name=Rajan&age=20

But nested objects are NOT allowed:

```
{
  "user": {
    "name": "Rajan",
    "role": "admin"
  }
}
```

For nested objects-→ extended: true
Uses the qs library (a more powerful parser).
Supports nested objects and arrays.
Example:
Request body:
user[name]=Rajan&user[role]=admin
This becomes:

```
{
  "user": {
    "name": "Rajan",
    "role": "admin"
  }
}
```