

NodeJS Assignment-1 Built in Modules

Github-Link

1) In this coding challenge, your task is to create a package.json file for your project using the npm init command. The package.json file is essential for managing dependencies, scripts, and other project-related details.

Ans- Using npm init command i have created package.json file

```
{
  "name": "assignment",
  "version": "2.0.3",
  "description": "This is my 1st nodeJS assignment.",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "assignment",
    "node",
    "js",
    "pw",
    "skills",
    "question1"
  ],
  "author": "MrPkMehta",
  "license": "ISC"
}
```

```
PS P:\Node Js> cd assignments\assignment-1
PS P:\Node Js\assignments\assignment-1> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (assignment-1) assignment
version: (1.0.0) 2.0.3
description: This is my 1st nodeJS assignment.
entry point: (index.js)
test command:
git repository:
keywords: assignment, node js, pw skills, question1
author: MrPkMehta
license: (ISC)
About to write to P:\Node Js\Assignments\assignment-1\package.json:
```

2) In the same project directory created in the above assignment, your task is to create a new file index.js and using the fs module add information about Node.js architecture to a new file nodejs_architecture.txt.

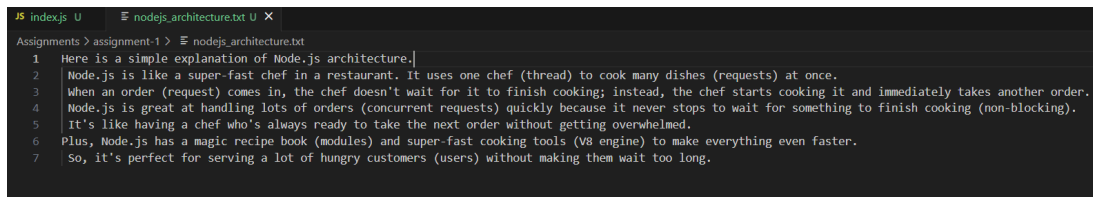
Ans-

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

const text = `Here is a simple explanation of Node.js architecture.\n Node.js is like a super-fast chef in a restaurant. It uses one chef (

fs.writeFile('nodejs_architecture.txt', text, function(err) {
  if (err) {
    console.log("Error in writing file.");
  } else {
    console.log("Writing in file is successful");
  }
});
```

Output



```
1 Here is a simple explanation of Node.js architecture.
2 Node.js is like a super-fast chef in a restaurant. It uses one chef (thread) to cook many dishes (requests) at once.
3 When an order (request) comes in, the chef doesn't wait for it to finish cooking; instead, the chef starts cooking it and immediately takes another order.
4 Node.js is great at handling lots of orders (concurrent requests) quickly because it never stops to wait for something to finish cooking (non-blocking).
5 It's like having a chef who's always ready to take the next order without getting overwhelmed.
6 Plus, Node.js has a magic recipe book (modules) and super-fast cooking tools (V8 engine) to make everything even faster.
7 So, it's perfect for serving a lot of hungry customers (users) without making them wait too long.
```

3) Continuing assignment 2. Here, let's create a new file named index.js and use the fs module to read the content of nodejs_architecture.txt and print the content to the console.

Ans-

```
//to read the content to the console
const fs = require('fs');

fs.readFile('nodejs_architecture.txt',function(err, data) {
  if (err) {
    console.log("Error in reading file");
  } else {
    console.log("Reading file is successful", data.toString());
  }
})
```

Output

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS P:\Node Js\assignments\assignment-1> node index1.js
Reading file is successful Here is a simple explanation of Node.js architecture.
Node.js is like a super-fast chef in a restaurant. It uses one chef (thread) to cook many dishes (requests) at once.
When an order (request) comes in, the chef doesn't wait for it to finish cooking; instead, the chef starts cooking it and immediately takes another order.
Node.js is great at handling lots of orders (concurrent requests) quickly because it never stops to wait for something to finish cooking (non-blocking).
It's like having a chef who's always ready to take the next order without getting overwhelmed.
Plus, Node.js has a magic recipe book (modules) and super-fast cooking tools (V8 engine) to make everything even faster.
So, it's perfect for serving a lot of hungry customers (users) without making them wait too long.
PS P:\Node Js\assignments\assignment-1> 
```

4) In this coding challenge, you will continue working with the file created in the previous assignments. Here your task is to access the existing `nodejs_architecture.txt` file and use the `fs` module to append additional data to it. Specifically, add some advantages of Node.js to the file and print the file content to the console.

Ans-

```
//to read and append to the content to the console
const fs = require('fs');

fs.readFile('nodejs_architecture.txt', function(err, data) {
  if (err) {
    console.log("Error in reading file");
  } else {
    console.log("Reading file is successful", data.toString());
  }
})

const text2 = `
Advantages of node js:
Node.js excels in speed, non-blocking I/O, and event-driven architecture, ideal for high-performance, real-time applications. It uses JavaScript.

fs.appendFile('nodejs_architecture.txt', text2, function(err) {
  if(err) {
    console.log("error in appending file");
  }
  else{
    console.log("success in appending file");
  }
})
```

Output

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS P:\Node Js\assignments\assignment-1> node index1.js
success in appending file
Reading file is successful Here is a simple explanation of Node.js architecture.
Node.js is like a super-fast chef in a restaurant. It uses one chef (thread) to cook many dishes (requests) at once.
When an order (request) comes in, the chef doesn't wait for it to finish cooking; instead, the chef starts cooking it and immediately takes another order.
Node.js is great at handling lots of orders (concurrent requests) quickly because it never stops to wait for something to finish cooking (non-blocking).
It's like having a chef who's always ready to take the next order without getting overwhelmed.
Plus, Node.js has a magic recipe book (modules) and super-fast cooking tools (V8 engine) to make everything even faster.
So, it's perfect for serving a lot of hungry customers (users) without making them wait too long.

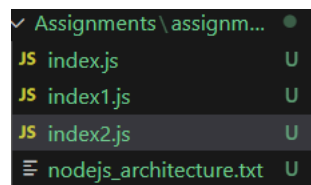
Advantages of node js:

Node.js excels in speed, non-blocking I/O, and event-driven architecture, ideal for high-performance, real-time applications. It uses JavaScript across the stack, has a vast package ecosystem, and scales effortlessly. Node.js simplifies development with code sharing and enjoys strong community support, making it a top choice for modern, high-performance web applications.
PS P:\Node Js\assignments\assignment-1> 
```

5) To wind up the fs module walk" through challenges, let's delete the nodejs_architecture.txt file. On deletion print "File Deleted Successfully" to the console.

Ans-

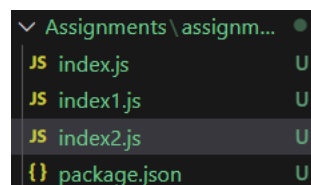
Before



```
//To delete the file
const fs = require('fs');

fs.unlink('nodejs_architecture.txt', function(err) {
  if (err) {
    console.log("Error in Deleting File");
  } else {
    console.log("File deleted successfully");
  }
})
```

After



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS P:\Node Js\assignments\assignment-1> node index2.js
File deleted successfully
PS P:\Node Js\assignments\assignment-1> █
```

6) Assume a situation where our server restricts access to its configuration via the user interface. The only way to obtain the OS and release information is through a programmatic approach. In this challenge, you are expected to use the `os` module and print the `os` name and the `os`-release version to the console.

Ans-

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

console.log("OS Name: " + os.hostname());

console.log("OS Release Version: " + os.release());
```

Output

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS P:\Node Js\assignments\assignment-1> node index3.js
OS Name: LAPTOP-5CAPSL3Q
OS Release Version: 10.0.25905
PS P:\Node Js\assignments\assignment-1> █
```

7) In this challenge, you are required to use Node.js and the built-in HTTP module to create a server that displays the text "I Am Happy To Learn Full Stack Web Development From PW Skills!" on the browser screen.

The goal is to utilize the HTTP module to create an HTTP server, set the port, and appropriate content type, and send the message as a response to the client's request, allowing it to display on the browser.

Ans-

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

```
const server = http.createServer((req, res) => {
  if (req.url == '/') {
    res.write('<h1>I Am Happy To Learn Full Stack Web Development From PW Skills! <h1>')
  }
  res.end();
});

server.listen(5004);
console.log("I am Using port 5004 to run the http server");
```

Output

I Am Happy To Learn Full Stack Web Development From PW Skills!

8) Let's simulate a subscription feature similar to YouTube. Using the events module, we'll create a custom event named "subscribe". When this event is triggered, it should display a message in the console indicating that the user has subscribed.

Ans-

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

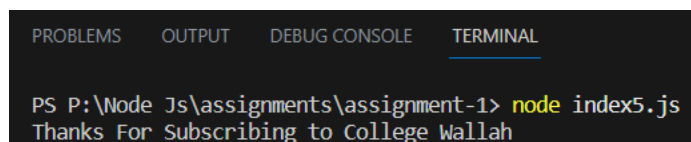
const eventEmitter = new EventEmitter();

const subscribeMessage = (channelName) => {
  console.log(`Thanks For Subscribing to ${channelName}`);
};

eventEmitter.addListener("subscribe", subscribeMessage);

eventEmitter.emit("subscribe", "College Wallah");
```

Output



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS P:\Node Js\assignments\assignment-1> node index5.js
Thanks For Subscribing to College Wallah
```

9) While working with the events module, one interesting observation is that when an event is created and called, the associated event handler is triggered. However, what happens if we remove an event and then try to call it? In this coding challenge let's create an event handler and call it. Later let's remove the event handler and observe what happens when we call it.

Ans -

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

const eventEmitter = new EventEmitter();

const subscribeMessage = (channelName) => {
  console.log(`Thanks For Subscribing to ${channelName}`);
};

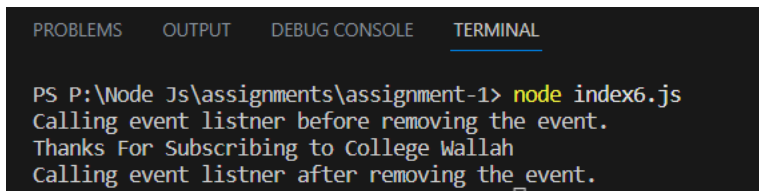
eventEmitter.addListener("subscribe", subscribeMessage);

console.log("Calling event listner before removing the event.");
eventEmitter.emit("subscribe", "College Wallah");

console.log("Calling event listner after removing the event.");
eventEmitter.removeListener("subscribe", subscribeMessage);

eventEmitter.emit("subscribe", "College Wallah");
```

Output



```
PS P:\Node Js\assignments\assignment-1> node index6.js
Calling event listner before removing the event.
Thanks For Subscribing to College Wallah
Calling event listner after removing the event.
```

here we can see that after removing the event handler the event function does not execute to the console.

10) In continuation of the 8th question, let's now explore the concept of the maximum number of listeners allowed for event handlers. For this coding challenge, your task is to determine the current maximum number of event listeners associated with an event and then set the maximum number of event listeners to 5. Note that the default maximum number of listeners might vary. Your task is to limit the number of listeners to 5.

Ans-

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

const eventEmitter = new EventEmitter();

const subscribeMessage = (channelName) => {
  console.log(`Thanks For Subscribing to ${channelName}`);
};

eventEmitter.addListener("subscribe", subscribeMessage);

eventEmitter.emit("subscribe", "College Wallah");

console.log(
  `The default maximum number of event listners are: ${eventEmitter.getMaxListeners()}`
);

eventEmitter.setMaxListeners(5);

console.log(
```

```
`The updated maximum number of event listeners are: ${eventEmitter.getMaxListeners()}`  
);
```

Output

```
PS P:\Node Js\assignments\assignment-1> node index7.js  
Thanks For Subscribing to College Wallah  
The default maximum number of event listeners are: 10  
The updated maximum number of event listeners are: 5  
PS P:\Node Js\assignments\assignment-1> █
```