

Node.js NPM

- NPM is a package manager for Node.js packages, or modules if you like. Modules are JavaScript libraries you can include in your project.
- www.npmjs.com hosts thousands of free packages to download and use.
- The NPM program is installed on your computer when you install Node.js

Download a Package

- Open the command line interface and tell NPM to download the package you want.

```
C:\Users\Your Name>npm install upper-case
```

- NPM creates a folder named "node_modules", where the package will be placed. All packages you install in the future will be placed in this folder.

Using a Package

```
var http = require('http');  
var uc = require('upper-case');  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.write(uc.upperCase("Hello Vijayan!"));  
  res.end();  
}).listen(8080);
```

```
Node.js command prompt - node node_npm_uppercase.js
Your environment has been set up for using Node.js 10.16.3 (x64) and npm.

C:\Users\Admin>d:

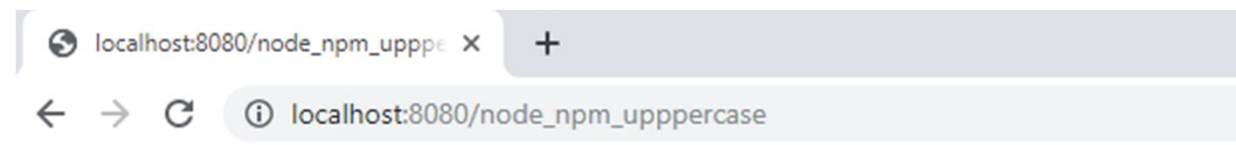
D:\>cd D:\..\2019-20 Fall Sem 2019-20\ITE1002\module-5\nodervi

D:\..\2019-20 Fall Sem 2019-20\ITE1002\module-5\nodervi>npm install upper-case
npm WARN saveError ENOENT: no such file or directory, open 'D:\..\2019-20 Fall Sem 2019-20\ITE1002\module-5\nodervi\package.json'
npm WARN enoent ENOENT: no such file or directory, open 'D:\..\2019-20 Fall Sem 2019-20\ITE1002\module-5\nodervi\package.json'
npm WARN nodervi No description
npm WARN nodervi No repository field.
npm WARN nodervi No README data
npm WARN nodervi No license field.

+ upper-case@1.1.3
updated 1 package and audited 287 packages in 2.45s
found 0 vulnerabilities

D:\..\2019-20 Fall Sem 2019-20\ITE1002\module-5\nodervi>node_npm_uppercase.js

D:\..\2019-20 Fall Sem 2019-20\ITE1002\module-5\nodervi>node node_npm_uppercase.js
```



HELLO VIJAYAN!

Node JS - Events

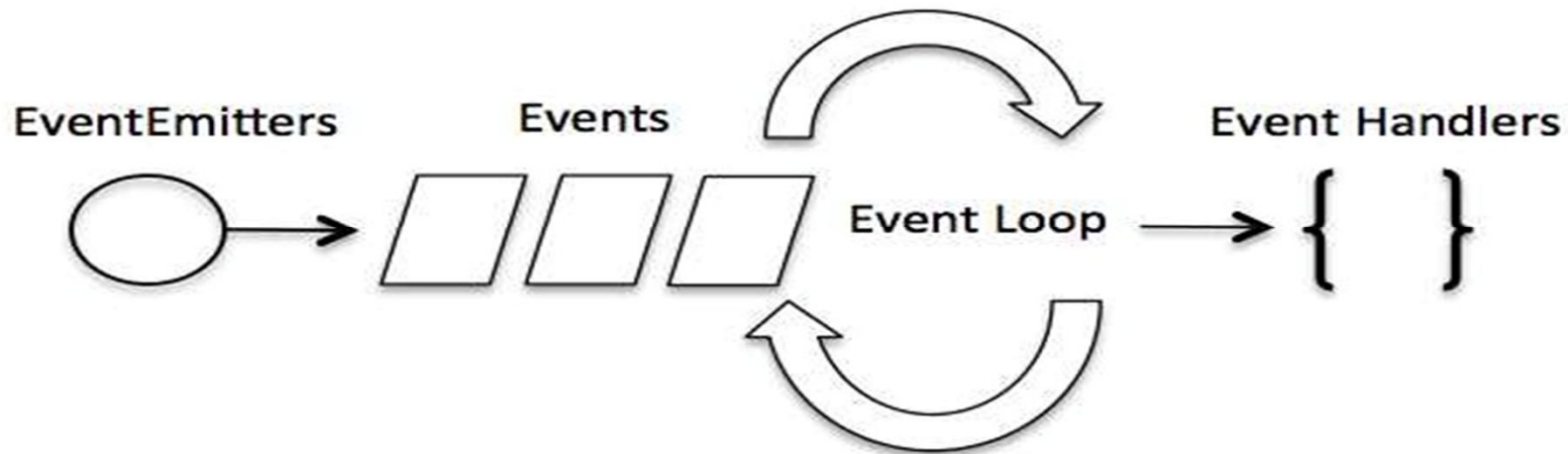
- Node.js is perfect for event-driven applications.
- Every action on a computer is an event. Like when a connection is made or a file is opened.
- Objects in Node.js can fire events, like the **readStream object fires events when opening and closing a file.**

```
C:\Users\admin\node_event_fileopen.js
```

```
var fs = require('fs');  
var readStream = fs.createReadStream('./readfile1.html');  
readStream.on('open', function () {  
  console.log('The file is open');});
```

Event-Driven Programming

- Node.js uses events heavily and it is also one of the reasons why Node.js is pretty **fast** compared to other similar technologies. As soon as **Node starts its server, it simply initiates its variables, declares functions and then simply waits for the event to occur.**
- In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected.



What and Why event emitter?

- Put simply, it allows you to **listen for "events" and assign actions** to run when those events occur. If you're familiar with front-end JavaScript, you'll know about mouse and keyboard events that occur on certain user interactions. These are very similar, except that we can **emit events on our own, when we want to**, and not necessary based on user interaction.
- The principles EventEmitter is based on have been called the **publish/subscribe model**, because we can subscribe to events and then publish them. There are many front-end libraries built with pub/sub support, but Node has it build in.

- Node.js has a built-in module, called "Events", where *you can create-, fire-, and listen for- your own events.*
- All event properties and methods are an instance of an **EventEmitter** object.
- You can assign event handlers to your own events with the EventEmitter object. To fire an event, use the **emit()** method.

```
var events = require("events");  
var myEmitter=new events.EventEmitter();  
myEmitter.on("someEvent",function () {  
    console.log("event has occurred");  });  
myEmitter.emit("someEvent");
```

Node.js command prompt

```
D:\..2019-20 Fall Sem 2019-20\ITE1002\module-5\nodervi>node node_event_emitter.js  
event has occurred  
D:\..2019-20 Fall Sem 2019-20\ITE1002\module-5\nodervi>
```

- `// Import events module`
`var events = require('events');`
- Create an `eventEmitter` object
`var eventEmitter = new events.EventEmitter();`
- Bind event and event handler as follows
`eventEmitter.on('eventName', eventHandler);`
- `// Fire an event`
`eventEmitter.emit('eventName');`

EventEmitter properties and methods

Method	Description
addListener(event, listener)	Adds the specified listener
defaultMaxListeners	Sets the maximum number of listeners allowed for one event. Default is 10
emit(event, [arg1], [arg2], [...])	Call all the listeners registered with the specified name
eventNames()	Returns an array containing all registered events
getMaxListeners()	Returns the maximum number of listeners allowed for one event
listenerCount(emitter, event)	Returns the number of listeners with the specified name
listeners(event)	Returns an array of listeners with the specified name
on(event, listener)	Adds the specified listener
once()	Adds the specified listener once. When the specified listener has been executed, the listener is removed

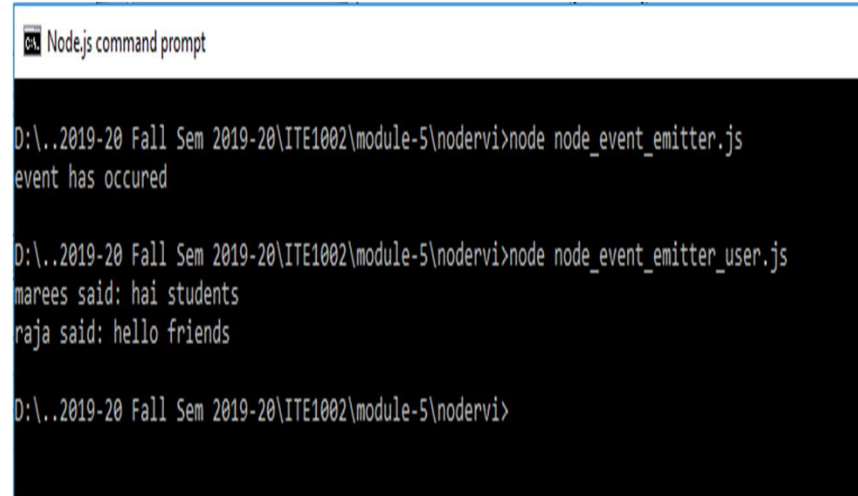
EventEmitter properties and methods

<code>prependListener()</code>	Adds the specified listener as the first event with the specified name
<code>prependOnceListener()</code>	Adds the specified listener as the first event with the specified name, once. When the specified listener has been executed, the listener is removed
<code>removeAllListeners([event])</code>	Removes all listeners with the specified name, or ALL listeners if no name is specified
<code>removeListener(event, listener)</code>	Removes the specified listener with the specified name
<code>setMaxListeners(n)</code>	Sets the maximum number of listeners allowed for one event. Default is 10

util module

- The util module is primarily designed to support the needs of Node.js' own internal APIs.
- However, many of the utilities are useful for application and module developers as well.
- **util.inherits(constructor, superConstructor)**
- Inherit the prototype methods from one constructor into another. The prototype of constructor will be set to a new object created from superConstructor.

```
var events=require('events');
var util=require('util');
var person=function(name){this.name=name;};
util.inherits(person,events.EventEmitter);
var vijay=new person('vijay');
var raja=new person('raja');
var shri=new person('shri');
var people=[vijay,raja,shri];
people.forEach(function(person){
    person.on('speak',function(msg) {
        console.log(person.name+' said: '+msg);});});
vijay.emit('speak','hai students');
raja.emit('speak','hello friends');
```



```
Node.js command prompt
D:\..2019-20 Fall Sem 2019-20\ITE1002\module-5\nodervi>node node_event_emitter.js
event has occurred
D:\..2019-20 Fall Sem 2019-20\ITE1002\module-5\nodervi>node node_event_emitter_user.js
marees said: hai students
raja said: hello friends
D:\..2019-20 Fall Sem 2019-20\ITE1002\module-5\nodervi>
```

Node.js VS Apache

1. It's fast
2. It can handle tons of concurrent requests
3. It's written in JavaScript (which means you can use the same code server side and client side)

Platform	Number of request per second
PHP (via Apache)	3187,27
Static (via Apache)	2966,51
Node.js	5569,30

Callback

- Callback is an **asynchronous equivalent** for a function
- A callback function **is called at the completion of a given task.**
- Node **makes heavy use of callbacks.**
- All the APIs of Node are written in such a way that they support callbacks.

Callback

- For example, a function to read a file may start reading file and return the control to the execution environment immediately so that the next instruction can be executed.
- Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as a parameter.
- So there is no blocking or wait for File I/O.
- This makes Node.js highly scalable, as it can process a high number of requests without waiting for any function to return results.

Blocking Code Example

```
var fs = require("fs");  
var data = fs.readFileSync('input.txt');  
console.log(data.toString()); console.log("Program  
Ended");
```

o/p:

- Welcome to node.js
- Program Ended

Non-Blocking Code Example

```
var fs = require("fs");  
fs.readFile('input.txt', function (err, data) { if (err) return  
console.error(err); console.log(data.toString()); });  
console.log("Program Ended");
```

o/p:

- Program Ended
- Welcome to node.js

Blocking and Non Blocking code

- These two examples explain the concept of blocking and non-blocking calls.
- The first example shows that the **program blocks until it reads the file and then only it proceeds to end the program.**
- The second example shows that the program **does not wait for file reading and proceeds to print "Program Ended" and at the same time, the program without blocking continues reading the file.**

Blocking and Non Blocking code

- Thus a **blocking program executes very much in sequence.**
- From the programming point of view, it is **easier to implement the logic but non-blocking programs do not execute in sequence.**
- In case a **program needs to use any data to be processed, it should be kept within the same block to make it sequential execution.**

Conclusion

- Node.js faster than apache but it more hungry system's CPU and memory
- Node.js use event based programming, it make the server doesn't wait for the IO operation to complete while it can handle other request at the same time
- Ref: <https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js>