## **List down the major benefits of using Node.js?**

***Fast:*** Node.js is built on Google Chrome’s V8 JavaScript Engine which makes its library very fast in code execution

***Asynchronous:***  Node.js based server never waits for an API to return data thus making it asynchronous

***Scalable:***  It is highly scalable because of its event mechanism which helps the server to respond in a non-blocking way

***Open Source*** : Node.js has an extensive open source community which has contributed in producing some excellent modules to add additional capabilities to Node.js applications

***No Buffering:*** Node.js applications simply output the data in chunks and never buffer any data

## **Why Node.js is single threaded?**

Node.js uses a single threaded model in order to support async processing. With async processing, an application can perform better and is more scalable under web loads. Thus, Node.js makes use of a [single-threaded model](https://www.edureka.co/blog/nodejs-tutorial/#nodearchitecture) approach rather than typical thread-based implementation.

## **How do Node.js works?**

Node.js is a virtual machine that uses JavaScript as its scripting language and runs on a v8 environment. It works on a single-threaded event loop and a non-blocking I/O which provides high rate as it can handle a higher number of concurrent requests. Also, by making use of the [‘HTTP’ module](https://www.edureka.co/blog/nodejs-tutorial/#http), Node.js can run on any stand-alone web server.

## **What is the difference between Asynchronous and Non-blocking?**

Asynchronous means not synchronous. Using these we can make asynchronous HTTP requests that do not wait for the server to respond. These functions continue to respond to the request for which it has already received the server response.

Non-blocking functions are used in regards with I/O operations. They immediately respond with whatever data is available and keeps on running as per the requests. In case, any answer couldn’t be retrieved then the API returns immediately with an error.

## **What is package.json?**

The [package.json file](https://www.edureka.co/blog/nodejs-tutorial/#json) in Node.js is the heart of the entire application. It is basically the manifest file that contains the metadata of the project where we define the properties of a package.

## **What do you understand by Event-driven programming?**

Event-driven programming is a programming approach that heavily makes use of events for triggering various functions. An event can be anything like a mouse click, key press, etc. When an event occurs, a call back function is executed that is already registered with the element. This approach mainly follows the publish-subscribe pattern. Because of [event-driven programming](https://www.edureka.co/blog/nodejs-tutorial/#events), Node.js is faster when compared to other technologies.

## **What is an Event loop in Node.js and how does it work?**

An event loop in Node.js handles all the asynchronous callbacks in an application. It is one of the most important aspects of Node.js and the reason behind Node.js have non-blocking I/O. Since Node.js is an event-driven language, you can easily attach a listener to an event and then when the event occurs the callback will be executed by the specific listener. Whenever functions like setTimeout, http.get, and fs.readFile are called, Node.js executed the event loop and then proceeds with the further code without waiting for the output. Once the entire operation is finished, Node.js receives the output and then executes the callback function. This is why all the callback functions are placed in a queue in a loop. Once the response is received, they are executed one by one.

## **Explain REPL in the context of Node.js.**

REPL in Node.js stands for **R**ead, **E**val, **P**rint, and **L**oop. It represents a computer environment such as a window console or Unix/Linux shell where any command can be entered and then the system can respond with an output. Node.js comes bundled with a REPL environment by default. REPL can perform the below-listed tasks:

* Read: Reads the user’s input, parses it into JavaScript data-structure and then stores it in the memory.
* Eval: Receives and evaluates the data structure.
* Print: Prints the final result.
* Loop: Loops the provided command until *CTRL+C* is pressed twice.

## **What is an error-first callback in Node.js?**

Error-first callbacks in Node.js are used to pass errors and data. The very first parameter you need to pass to these functions has to be an error object while the other parameters represent the associated data. Thus you can pass the error object for checking if anything is wrong and handle it. In case there is no issue, you can just go ahead and with the subsequent arguments.

var myPost = new Post({title: 'edureka'});

myPost.save(function(err,myInstance){

if(err){

//handle error and return

}

//go ahead with `myInstance`

});

## **Explain the purpose of module.exports?**

A module in Node.js is used to encapsulate all the related codes into a single unit of code which can be interpreted by shifting all related functions into a single file. module.exports provide two functions which can be imported in another file.

## **What do you understand by Reactor Pattern in Node.js?**

**Reactor Pattern** in Node.js is basically a concept of non-blocking I/O operations. This pattern provides a handler that is associated with each I/O operation and as soon as an I/O request is generated, it is then submitted to a *demultiplexer*. This demultiplexer is a notification interface which is capable of handling concurrency in non-blocking I/O mode. It also helps in collecting each and every request in the form of an event and then place each event in a queue. Thus resulting in the generation of the Event Queue. Simultaneously, we have our event loop which iterates the events present in the Event Queue.

## **What do you understand by callback hell?**

Callback Hell is also known as the Pyramid of Doom. It is a pattern caused by intensively nested callbacks which are unreadable and unwieldy. It typically contains multiple nested callback functions which in turn make the code hard to read and debug. It is caused by improper implementation of the asynchronous logic.

async\_A(function(){

async\_B(function(){

async\_C(function(){

async\_D(function(){

....

});

});

});

});

## **Explain the concept of middleware in Node.js?**

In general, middleware is a function receives the Request and Response objects. In other words, in an application’s request-response cycle these functions have access to various request & response objects along with the next function of the cycle. The next function of middleware is represented with the help of a variable, usually named next. Most commonly performed tasks by the middleware functions are:

* Execute any type of code
* Update or modify the request and the response objects
* Finish the request-response cycle
* Invoke the next middleware in the stack