# Day 1 NodeJs Introduction

## What is NodeJS

JavaScript is primarily used in web pages and gaming applications, often in conjunction with other programming languages.

Node JS is a runtime environment for executing JavaScript code outside of a web browser. It allows for server-side programming. Node JS is popular in the industry for its ability to create cross-platform applications compatible with Android and iOS devices.

## More on NodeJS

Node.js is an asynchronous, event-driven JavaScript runtime designed for building scalable network applications. It **handles multiple connections** concurrently and **sleeps when there's no work to be done**, contrasting with thread-based networking which is less efficient. Node.js users don't have to worry about process deadlocks since there are no locks.

Node.js is influenced by systems like Ruby's Event Machine or Python's Twisted, but it takes the event model further by presenting an event loop as a runtime construct. It doesn't require a blocking call to start the event loop. HTTP is a primary focus in Node.js, designed for streaming and low latency.

Although Node.js doesn't use threads, it can still utilize multiple cores. Child processes can be spawned, and the cluster module allows for socket sharing to enable load balancing over cores.

Node.js is built against modern versions of the V8 engine, ensuring timely delivery of new ECMAScript features. These features are split into three groups: shipping (default), staged (require a runtime flag), and in progress (discouraged for production use).

To find out which features are available in each Node.js release, you can use the --v8-options argument and grep for in-progress features, but be cautious as they may be incomplete or broken. The V8 team works on optimizing new language features to reach performance parity with their ES5 counterparts, tracked on the six-speed website.

The V8 team also coordinates improvement efforts through a performance plan, addressing areas that need enhancement and providing design documents for solutions.

## Use Cases

**Building Utilities with Your Machine:** Node.js can be used to create various utilities that automate tasks on your machine. This can include tasks related to managing files, performing automated backups, or even implementing live-reloading functionality for web development.

Sample Code (Live Reload Utility)

// Example of a live reload utility using Node.js

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

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

**fs.watch('path\_to\_watch\_directory', (event, filename) => {**

**if (event === 'change' && filename.endsWith('.html')) {**

**console.log(`Reloading ${filename}...`);**

// Implement code to trigger live reload in your application

**}**

**});**

**http.createServer((req, res) => {**

**res.write('Live Reload Utility is running!');**

**res.end();**

**}).listen(3000);**

**Web Application with a Web Server:** Node.js is commonly used to build web applications. It can serve as the backend server that handles HTTP requests and communicates with databases. Popular frameworks like Express.js provide a powerful platform for building web applications in Node.js.

Sample Code (Simple Webserver)

**var http = require(‘http’);**

**var server = http.createServer(function(request, response) {**

**console.log(‘Got a request!’);**

**response.write(‘hi’);**

**response.end();**

**});**

**Server.listen(3000);**

## Modules

**How to Load One File into Another:** Node.js allows you to modularize your code by breaking it into smaller, reusable files. You can use the **require** function to load these files into other files.

Sample Code (Loading a Module):

// File: myModule.js

**module.exports = () => {**

**console.log('This is my module!');**

**};**

// File: main.js

**const myModule = require('./myModule');**

**myModule();**

NPM Modules for Managing Packages: npm (Node Package Manager) is a tool that comes with Node.js, allowing you to install and manage external packages (libraries) for your projects. These packages can provide various functionalities and help you avoid reinventing the wheel.

// In your terminal, run:

**npm install packageName**

// In your code:

**const packageName = require('packageName');**

**Node Init for Managing Dependencies:** npm init is a command that helps you initialize a new Node.js project. It creates a package.json file which includes information about your project and its dependencies. This makes it easier to manage dependencies.

// In your terminal, run:

**npm init**

// Follow the prompts to create your package.json file

**Keeping Dependencies Updated:** npm allows you to easily update your project's dependencies to their latest versions. This ensures you're using the most up-to-date and secure packages.

// In your terminal, run:

**npm update**

// This will update your project's dependencies to their latest compatible versions

## Quick Quiz

What is Node JS?

**Node.js is an open source server environment**

How is Node JS initiated on a computer?

**Through the command line interface**

Why do we use Node JS?

**Node JS is asynchronous**

What can Node JS do?

**Node JS can send dynamic content   
Node JS contains some tasks that can be executed on certain events eg someone trying to access a port on the server**

What is a module in Node JS the same as in JavaScript?

**Libraries.**

What is NPM?

**Node JS Package Manager**

What is contained  in a Node JS Package?

**A package in Node.js contains all the files you need for a module**