**Node.js**

1. **What is Node.js?**

Node.js is an open-source, server-side runtime environment that allows JavaScript to run outside the browser. It is built on Chrome's V8 JavaScript engine.

**Purpose:**

* It helps in building fast, scalable, and efficient network applications.
* Unlike traditional server-side languages (PHP, Java, etc.), it handles multiple requests simultaneously.

**Usage:**

* Used for backend development, API creation, real-time applications (chat apps, stock trading), and microservices.
* Ideal for handling high I/O operations like file handling, database interactions, and networking.

**Description:**

Node.js is a **single-threaded, event-driven, non-blocking I/O** environment.

* **Single-threaded:** Uses one thread to handle multiple tasks.
* **Event-driven:** Works based on events and callbacks.
* **Non-blocking I/O:** Doesn't wait for one task to complete before starting another.

**How We Use It:**

We install Node.js on a machine and use it to execute JavaScript files outside the browser. Developers use it to build web servers, process data, and interact with databases.

1. **Node.js Architecture**

**Definition:**

Node.js architecture follows a **Single-Threaded Event Loop model** to handle multiple client requests efficiently without blocking operations.

**Purpose:**

* Ensures **high performance** with minimal resources.
* Handles multiple requests **asynchronously** without creating multiple threads.
* Used for **real-time applications** like chats, streaming services, and IoT.

**Usage:**

* Web servers (handling multiple users at once).
* Streaming platforms like **Hotstar**, **YouTube**, and **Netflix**.
* Chat applications like **WhatsApp Web** or **Slack**.

**Description:**

Node.js architecture consists of:

1. **Single Thread:** Unlike traditional servers (PHP, Java) that create a new thread per request, Node.js uses **one thread for all requests**.
2. **Non-blocking I/O:** It doesn’t wait for one task to finish; it starts another task while waiting.
3. **Event Loop:** Keeps checking for new tasks and executes them when ready.
4. **Callback Mechanism:** When a task (like reading a file) is completed, a callback function is executed.

**How We Use It:**

When a request (like fetching user details from a database) comes in:

1. The request goes to the **Single Thread**.
2. The **Event Loop** checks if the request is non-blocking (like database calls).
3. If non-blocking, it is sent to the **Worker Threads** to process.
4. Once the task is complete, the response is sent back to the user.
5. **How Node.js Works**

**Definition:**

Node.js works using a **Single-Threaded Event Loop Model**, which enables it to handle multiple client requests asynchronously without blocking operations.

**Purpose:**

* To **handle multiple requests efficiently** without creating multiple threads.
* To make applications **fast and scalable** by using non-blocking I/O operations.
* To support **real-time applications** like chat apps, stock market dashboards, and streaming platforms.

**Usage:**

* Building web servers
* Handling file system operations
* Creating real-time applications like **Zomato order tracking**
* Streaming services like **JioCinema and Hotstar**

**Description:**

Node.js follows these steps when handling a request:

1. **Client sends a request** (e.g., fetching user data).
2. **Node.js processes the request using the event loop**.
3. If the request is:
   * **Non-blocking (like fetching from a database)** → It is handled asynchronously in the background.
   * **Blocking (like complex calculations)** → It is processed by worker threads.
4. **Once completed, the response is sent back to the client** without blocking other requests.

**How We Use It:**

When a request is made (e.g., fetching user data from a database), Node.js does not wait for the response. Instead, it moves on to the next task while the database operation completes in the background.

1. **Installing Node.js**

**Definition:**

Installing Node.js means setting up the Node runtime environment on your system so that you can execute JavaScript outside a web browser.

**Purpose:**

* To enable JavaScript to run on the server-side.
* To allow developers to use npm (Node Package Manager) for installing libraries.
* To develop and test Node.js applications locally.

**Usage:**

* Required for building web servers, APIs, and real-time applications.
* Used in backend development for platforms like Flipkart, Swiggy, and Ola.

**Description:**

Node.js installation includes:

1. Node.js runtime (to execute JavaScript code).
2. npm (Node Package Manager) (to install third-party modules).

**How We Use It:**

We download and install Node.js from its official website and use the terminal to verify the installation.

**Installation Steps (Windows & macOS):**

1. Go to <https://nodejs.org>.
2. Download the LTS (Long-Term Support) version.
3. Run the installer and follow the instructions.

**5. Your First Node Program**

**Definition:**

A simple script written in JavaScript that runs in the Node.js environment.

**Purpose:**

* To verify that Node.js is installed correctly.
* To understand how JavaScript runs outside the browser.

**Usage:**

* Used to execute JavaScript code on the backend.
* Helps test whether the Node.js environment is working.

**Description:**

A Node.js program is written in JavaScript and saved as a .js file. It is executed using the node command.

**How We Use It:**

We create a .js file and run it using the Node.js runtime.

**6. Node Module System**

**Definition:**

The Node.js Module System allows us to **organize code into reusable files** called modules.

**Purpose:**

* To avoid writing the same code multiple times.
* To divide large applications into smaller, manageable parts.
* To use built-in and third-party modules for common tasks.

**Usage:**

* Used for structuring Node.js applications efficiently.
* Helps in creating reusable code for **database operations, authentication, and utilities**.

**Description:**

Node.js supports three types of modules:

1. **Built-in Modules** (like fs, http, os).
2. **User-defined Modules** (custom modules created by developers).
3. **Third-party Modules** (installed via npm, like express).

**How We Use It:**

We use the require() function to import modules.

**7. Global Object**

**Definition:**

The Global Object in Node.js provides built-in functions and variables accessible anywhere in a program.

**Purpose:**

* To provide utility functions like console.log(), setTimeout(), and process.
* To avoid unnecessary imports for commonly used features.

**Usage:**

* Used for logging messages, handling timers, and accessing system info.
* console.log() for logging messages
* setTimeout() for delayed execution
* setInterval() for repeated execution

**Description:**

Unlike the window object in browsers, Node.js has a global object. Some commonly used global properties:

* console.log() – Prints output to the console.
* setTimeout() – Runs code after a delay.
* \_\_dirname – Returns the directory path of the current file.
* \_\_filename – Get current file name

**How We Use It:**

We directly call global functions without requiring any module.

**8. Modules**

**Definition:**

Modules in Node.js are reusable pieces of code that can be imported and exported between files.

**Purpose:**

* To **keep code organized and modular**.
* To **enable code reusability**.

**Usage:**

* Used in **creating APIs, file operations, and handling database queries**.

**Description:**

Modules can be:

1. **Core Modules** – Built-in (fs, http).
2. **User-Defined Modules** – Created by developers.
3. **Third-Party Modules** – Installed using npm (express, mongoose).

**How We Use It:**

1. **Create a module:** Write reusable functions in a separate file.
2. **Export the module:** Use module.exports.
3. **Import the module:** Use require().

**9. Creating a Module**

**Definition:**

A **module** in Node.js is a separate file that contains reusable code, which can be imported and used in other files.

**Purpose:**

* To organize code into small, manageable pieces.
* To avoid repetition and improve **reusability**.
* To keep related functionality together.

**Usage:**

* Creating custom **utility functions** (e.g., logging, validation).
* Structuring projects by splitting features into different modules.
* Making code **easier to debug and maintain**.

**Description:**

A module in Node.js is simply a JavaScript file with functions or objects that we **export** and then **import** in another file.

**How We Use It:**

1. **Create a new JavaScript file (mathOperations.js)** and write some functions.
2. **Export the functions** using module.exports.
3. **Import the module in another file (app.js)** using require().

**10. Loading a Module**

**Definition:**

Loading a module in Node.js means importing the code from another file using the require() function.

**Purpose:**

* To **reuse existing functions** in different files.
* To make the code **modular and maintainable**.

**Usage:**

* Importing **built-in modules** (fs, os, http).
* Importing **user-defined modules** (./mathOperations).
* Importing **third-party modules** (express, lodash).

**Description:**

To use a module, we:

1. Call require() and specify the module name or file path.
2. Store the imported functions in a variable.
3. Use the functions in our script.

**How We Use It:**

* Built-in modules: const os = require("os");
* Custom modules: const math = require("./mathOperations");
* npm packages: const express = require("express");

**11. Module Wrapper Function**

**Definition:**

The **Module Wrapper Function** in Node.js wraps every module in a special function before execution.

**Purpose:**

* Provides a **private scope** for each module.
* Prevents **variable name conflicts** between modules.
* Automatically gives access to important variables like exports, require, module, \_\_dirname, and \_\_filename.

**Usage:**

* Helps maintain **modular, conflict-free code**.
* Allows Node.js to pass essential parameters to every module.

**Description:**

Each module in Node.js is wrapped inside this function:

(function (exports, require, module, \_\_filename, \_\_dirname) {

// Module code here

});

This ensures that variables inside a module **do not leak** into other files.

**How We Use It:**

The wrapper function is **automatically applied** to every module, so we don’t need to call it manually.

**12. Path Module**

**Definition:**

The path module in Node.js provides utilities for working with file and directory paths.

**Purpose:**

* Helps manage **file paths dynamically** across different operating systems.
* Allows us to extract **file names, extensions, and directories** easily.

**Usage:**

* Used in **file handling applications**.
* Helps in dynamically **constructing file paths**.

**Description:**

The path module provides methods like:

* path.basename(path) – Returns file name.
* path.dirname(path) – Returns directory name.
* path.extname(path) – Returns file extension.
* path.join(...paths) – Joins multiple path segments.

**How We Use It:**

We first **import the module** and then call its functions.

**13. OS Module**

**Definition:**

The os module in Node.js provides utilities to interact with the operating system and retrieve system-related information.

**Purpose:**

* Helps in **getting system details** such as CPU, memory, uptime, and network interfaces.
* Useful for **monitoring and optimizing applications** based on system resources.

**Usage:**

* Checking **available system memory** before running resource-intensive processes.
* Fetching **CPU details** to optimize performance.
* Getting the **OS type and platform** to adjust system-specific configurations.

**Description:**

The os module includes methods to fetch system details like:

* os.arch() – CPU architecture.
* os.cpus() – CPU information.
* os.freemem() – Available free memory.
* os.totalmem() – Total memory in the system.
* os.platform() – OS platform (linux, win32, darwin).
* os.uptime() – System uptime in seconds.

**How We Use It:**

We first **import the module** and then call its methods to get system information.

**14. File System Module (fs)**

**Definition:**

The fs module in Node.js allows us to work with the **file system**, enabling us to read, write, update, and delete files.

**Purpose:**

* To **read or write files** dynamically.
* To create, modify, or delete files and directories programmatically.
* Used in **logging, configuration management, and data storage**.

**Usage:**

* Reading files (fs.readFile) to **fetch stored data**.
* Writing to files (fs.writeFile) to **store logs or user data**.
* Deleting files (fs.unlink) to **remove outdated data**.

**Description:**

The fs module provides both **synchronous** (fs.readFileSync) and **asynchronous** (fs.readFile) methods for file operations.

**How We Use It:**

1. Reading a File
2. Writing to a File
3. Appending Data to a File
4. Deleting a File

**15. Events Module**

**Definition:**

The events module in Node.js allows us to create and handle **custom events** asynchronously.

**Purpose:**

* Helps in building **event-driven applications** like chat apps and live updates.
* Allows modules to **communicate** with each other using events.

**Usage:**

* Used in **server-side event handling** (e.g., when a user connects or sends a message).
* Implemented in **real-time applications** like stock price updates.

**Description:**

The events module provides the EventEmitter class, which allows us to:

1. **Create an event emitter**
2. **Register event listeners**
3. **Trigger events**

**How We Use It:**

* First, we **import** the events module.
* Then, we **create an event emitter**.
* We **listen** for an event using on().
* We **emit** the event using emit().

**16. Event Arguments**

**Definition:**

Event arguments allow us to **pass data** when an event is triggered.

**Purpose:**

* Helps in sending **dynamic information** with an event.
* Makes event-driven programs **more flexible**.

**Usage:**

* Used in **chat applications** (e.g., passing messages).
* Used in **logging systems** (e.g., passing error details).

**17. Extending EventEmitter**

**Definition:**

Extending EventEmitter means **creating a custom class** that inherits from EventEmitter, allowing us to define our own event-driven logic.

**Purpose:**

* Helps in **creating modular, reusable event-based classes**.
* Useful when building **custom objects that emit and handle events** (e.g., user interactions, API calls).

**Usage:**

* Used in **custom logging systems**, where we emit log messages.
* Used in **real-time applications** (e.g., stock updates, live scores).

**Description:**

Instead of creating a standalone EventEmitter, we **extend it in a class**, making our objects event-driven.

**How We Use It:**

1. **Create a class** that extends EventEmitter.
2. **Define methods** that emit events.
3. **Create an instance** and listen for events.

**18. HTTP Module**

**Definition:**

The http module in Node.js is used to **create web servers and handle HTTP requests and responses**.

**Purpose:**

* Helps in **creating web servers** without additional dependencies.
* Used in building **REST APIs and web applications**.

**Usage:**

* Handling **user requests** (e.g., login, fetching data).
* Serving **web pages dynamically**.

**Description:**

The http module allows us to:

1. **Create a web server** (http.createServer()).
2. **Listen for client requests** on a specific port.
3. **Respond to requests** with data or HTML.

**How We Use It:**

* We **import** the http module.
* We **create a server** and define how it handles requests.
* We **start the server** on a port.