Introduction to Node.js

Q -1) What is Node JS?

Node.js is a popular JavaScript runtime framework used to build fast and scalable applications. It allows developers to use JavaScript for server-side programming.

- **History and Evolution :**

Node.js was created by **Ryan Dahl** in **2009**. Before Node.js, JavaScript was mostly used in browsers only. Ryan wanted to make a tool that could handle many requests at the same time without slowing down. That's how Node.js was born.

Over time, many developers started using it to build web servers, APIs, real-time apps, and more. Node.js became popular because it was fast, easy to use, and had a big community.

- **Architecture:**

Node.js is built on Chrome's V8 JavaScript engine, which makes it very fast. It uses a single-threaded and event-driven model.

Instead of creating many threads for each request (like other languages), Node.js uses non-blocking I/O, meaning it can handle many tasks at once without waiting.

- **Key Features**

1. **Fast** – Uses V8 engine which runs JavaScript quickly.
2. **Asynchronous and Non-blocking** – Can handle many tasks at the same time.
3. **Single-threaded** – Uses one thread but still handles many requests.
4. **Cross-platform** – Works on Windows, macOS, and Linux.
5. **NPM (Node Package Manager)** – Comes with thousands of ready-made packages.
6. **Community Support** – Has a large and active developer community.
7. **Real-time Apps** – Great for apps like chat or live updates.

Q- 2) Compare Node.js with traditional server-side technologies like PHP and Java.

Comparison: Node.js vs PHP vs Java

Feature / Point	Node.js	PHP	Java
Language Used	JavaScript	PHP	Java
Runtime Environment	V8 Engine (Chrome)	Zend Engine or HHVM	Java Virtual Machine (JVM)
Concurrency	Non-blocking, asynchronous (event-driven)	Blocking, synchronous (traditional)	Multi-threaded
Performance	Very fast for I/O tasks	Slower for I/O-heavy apps	Fast and reliable, good for large systems
Scalability	Highly scalable	Less scalable for real-time apps	Very scalable
Ease of Learning	Easy for JS developers	Easy for beginners	Harder than PHP and Node.js
Community Support	Large and growing	Very large and old community	Mature and professional support
Best Used For	Real-time apps (chat, games), APIs	Simple websites, CMS (WordPress, etc.)	Enterprise apps, banking, large systems
Package Manager	NPM (very rich)	Composer (less rich than NPM)	Maven / Gradle
Hosting	Needs setup (Node	Easy and cheap	Needs Java

Feature / Point	Node.js	PHP	Java
	environment)	hosting	server (Tomcat, etc.)

Summary

- **Node.js is best for fast, real-time apps like chat or streaming.**
- **PHP is good for websites and blogs (like WordPress).**
- **Java is powerful for large-scale, enterprise-level applications.**

Each has its own strengths, and the best choice depends on the project.

Q – 3) Describe the role of npm (Node Package Manager) in Node.js development. Discuss common commands used in npm.

Role of npm (Node Package Manager) in Node.js

npm is the **default package manager for Node.js**. It helps developers easily install, share, and manage reusable code (called **packages** or **modules**).

Why npm is important:

- Lets you install third-party packages like Express, React, etc.
- Manages versions and dependencies of packages.
- Makes it easy to reuse code instead of writing from scratch.
- Helps in creating and managing your own project's packages.

When you create a Node.js project, a file called package.json is used to store information about your project and its dependencies.

Common npm Commands

Command	Description
npm init	Creates a new package.json file for your project.
npm init -y	Creates package.json with default values (quick setup).
npm install <package>	Installs a package (e.g., npm install express).
npm install	Installs all dependencies listed in package.json.
npm install <package> --save	Adds the package to your dependencies (not needed in latest npm).
npm install <package> --save-dev	Adds the package as a development dependency.
npm uninstall <package>	Removes a package.
npm update	Updates all packages.
npm list	Lists installed packages.
npm run <script>	Runs a script defined in package.json (like npm run start).

Conclusion:

npm is a key tool in Node.js development. It saves time by letting you use ready-made modules and keeps your project organized. With just a few commands, you can manage your entire project easily.

Q – 4) Describe the module system in JavaScript, including CommonJS and ES Modules. Discuss their significance for code organization.

Module System in JavaScript

In JavaScript, a **module** is a file that contains code (functions, variables, classes, etc.) that can be **imported** and **exported** between files.

Modules help you:

- Keep code organized.
- Reuse code easily.
- Avoid global variables and name conflicts.

---

1. CommonJS Modules (CJS)

- Used mainly in **Node.js**.
- Synchronous (loads modules one at a time).
- Uses require() to import and module.exports to export.

Example: math.js

```
function add(a, b) {
  return a + b;
}
module.exports = add;
```

app.js

```
const add = require('./math');
console.log(add(2, 3)); // 5
```

---

2. ES Modules (ESM)

- Standard in **modern JavaScript (ES6+)**.
- Works in browsers and Node.js (with .mjs or "type": "module").
- Uses import and export.

Example:

math.js

```
export function add(a, b) {
  return a + b;
}
```

app.js

```
import { add } from './math.js';
console.log(add(2, 3)); // 5
```

Significance for Code Organization :

- **Cleaner Code**: Each module handles a specific task (like math, user, API).
- **Reusability**: Easily reuse code across different files or projects.
- **Maintainability**: Easier to manage and update code in small parts.
- **Avoid Conflicts**: Variables and functions are scoped to the module.

Conclusion :

Modules in JavaScript (CommonJS and ES Modules) are very useful for writing clean, organized, and maintainable code.
CommonJS is great for Node.js, while ES Modules are the future and work both in browsers and Node.js.

Q – 5) Discuss what APIs are and how JavaScript can interact with them, focusing on Fetch API for making network requests.

What is an API?

API stands for Application Programming Interface.
It allows two different software systems to communicate with each other.

In web development, APIs are often used to:

- **Get data from a server (like weather info, user data, etc.)**
- **Send data to a server (like form submission, login info, etc.)**

How JavaScript Interacts with APIs

JavaScript can make network requests (like GET, POST) to APIs using:

- **XMLHttpRequest (older way)**
- **Fetch API (modern and easier way)**
- **Axios (a third-party library)**

Fetch API – The Modern Way

The Fetch API is a built-in JavaScript feature that lets you make HTTP requests easily.

Syntax:

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => {
    console.log(data); // use the data
  })
  .catch(error => {
    console.error('Error:', error);
  });
```

How it works:

1. **`fetch()` sends a request to the API URL.**
2. **`.then()` gets the response and converts it to JSON.**
3. **Another `.then()` handles the data.**
4. **`.catch()` handles any errors (like no internet or server issues).**

Example: Get User Data

```
fetch('https://jsonplaceholder.typicode.com/users')
  .then(res => res.json())
  .then(users => {
    users.forEach(user => {
      console.log(user.name);
    });
  });
```

Why APIs + JavaScript Are Useful :

- **Makes apps dynamic (load data without reloading the page).**
- **Helps build single-page apps.**
- **Works well with front-end frameworks like React.**

Conclusion :

APIs allow your JavaScript app to talk to servers and get/send data. The Fetch API makes this easy and clean, helping you build modern, data-driven web applications.
