Node.js Interview Questions

1. What is Node.js, and how does it differ from traditional server-side languages like PHP or Java?
   Node.js is an open-source, server-side runtime environment for executing JavaScript. It differs from traditional server-side languages by using a non-blocking, event-driven architecture, making it highly efficient and well-suited for I/O-intensive tasks.

2. Explain the concept of non-blocking I/O in Node.js. How does it work?
   Non-blocking I/O allows Node.js to continue processing other tasks without waiting for I/O operations to complete. It works by using asynchronous functions and callbacks, allowing other code to run while waiting for I/O operations to finish.

3. What is an event loop in Node.js, and why is it essential for its non-blocking behavior?
   The event loop is a fundamental part of Node.js that manages asynchronous tasks. It continually checks the message queue for pending tasks, making Node.js non-blocking and highly efficient.

4. What is the Node Package Manager (NPM), and how is it used in Node.js development?
   NPM is the package manager for Node.js. It is used to install, manage, and share packages or modules in Node.js applications.

5. What are modules in Node.js, and how do you create and use them?
   Modules are reusable pieces of code in Node.js. You create them using `module.exports` and use them in other files with `require`.

6. What is callback hell, and how can it be mitigated in Node.js?
Callback hell refers to deeply nested callbacks that can make code hard to read and maintain. It can be mitigated using techniques like named functions, Promises, or async/await.

7. What is the 'require' function, and how is it used in Node.js?
The 'require' function is used to import modules in Node.js. It loads and returns the exported object of the requested module.

8. Explain the difference between 'require' and 'import' for loading modules in Node.js.
'require' is the CommonJS module system used in Node.js, while 'import' is an ES6 feature used in modern JavaScript. They serve similar purposes but have different syntax.

9. What are the core modules in Node.js, and how do you use them?
Core modules are built-in modules provided by Node.js. You can use them by simply requiring the module name, e.g., 'fs' for file system operations or 'http' for creating web servers.

10. How can you handle errors in Node.js applications?
Errors in Node.js can be handled using try-catch blocks, error event listeners, or by passing errors as the first argument in callbacks.

11. What is the purpose of the 'util' module in Node.js?
The 'util' module provides utility functions for working with JavaScript objects, such as `util.inherits` and `util.promisify`.

12. How can you achieve file I/O operations in Node.js?
File I/O operations can be performed using the 'fs' (file system) module, which provides methods for reading and writing files.

13. Explain the concept of streams in Node.js and provide examples of when to use them.

Streams in Node.js are used for reading or writing data in chunks, rather than loading it all at once. They are useful for large files, network protocols, and real-time data processing.

14. What is 'process' in Node.js, and how can it be used for managing a Node.js application?

'process' is a global object in Node.js that provides information about the current Node.js process and allows you to manage it. You can use it for tasks like environment variable access and process control.

Express.js Interview Questions

1. What is Express.js, and how does it relate to Node.js?
   Express.js is a web application framework for Node.js that simplifies the process of building web applications and APIs.

2. What are middleware functions in Express.js, and why are they important?
   Middleware functions are functions that have access to the request and response objects in Express. They are important for tasks like authentication, logging, and request processing.

3. How do you handle routing in Express.js?
   Routing in Express.js is achieved by defining routes using methods like `app.get()`, `app.post()`, etc., and specifying the route handlers for those routes.

4. Explain the difference between 'app.get()' and 'app.use()' in Express.js routing.
   `app.get()` is used for defining a route that handles GET requests, while `app.use()` is more general and can be used to attach middleware functions to routes.

5. What is the purpose of the 'body-parser' middleware in Express.js?
   'body-parser' is middleware that parses incoming request bodies to make the data accessible via `req.body`. It is commonly used for handling POST and PUT requests with JSON or URL-encoded data.

6. How can you serve static files (e.g., CSS, JavaScript) in an Express.js application?
   You can use the `express.static` middleware to serve static files from a specified directory.

7. What are template engines in Express.js, and how do they work?**
   Template engines allow you to dynamically generate HTML using data. Popular template engines for Express.js include EJS, Pug, and Handlebars.

8. Explain the concept of RESTful APIs and how they are implemented in Express.js.
   RESTful APIs follow a set of architectural principles. In Express.js, RESTful routes are typically designed to perform CRUD operations on resources, using HTTP methods (GET, POST, PUT, DELETE).

9. What is middleware chaining in Express.js, and why is it useful?
   Middleware chaining involves applying multiple middleware functions to a route. It is useful for handling multiple tasks in sequence, such as authentication and authorization.

10. How do you handle and validate request parameters in Express.js?
    You can access request parameters using `req.params`, `req.query`, or `req.body`. Validation can be performed using middleware or libraries like Joi or Express Validator.

11. What is a session in Express.js, and how can you implement session management?
    A session is a way to store data for a user across multiple requests. Express.js provides middleware like 'express-session' for session management.

12. What is the purpose of the 'response' object in Express.js, and how can you use it?
    - The 'response' object is used to send responses to clients. It can be used to set headers, send data, and handle responses, such as rendering HTML views.

13. How can you handle errors in Express.js applications?
    - Errors in Express.js can be handled by defining custom error middleware and using the `next` function with an error object.