Q1: What is webpack?

A1: Webpack is a popular open-source JavaScript module bundler. It takes various assets, such as JavaScript, CSS, and images, and transforms them into a format that can be efficiently loaded and executed by a web browser.

Q2: What is a bundle in webpack?

A2: A bundle in webpack is the output generated by the bundling process. It consists of a collection of files and dependencies that have been combined and optimized for delivery to the client's browser.

Q3: What is an “entry point” in webpack?

A3: An entry point in webpack is the starting point for building the dependency graph. It is the module or file that webpack uses to begin the bundling process.

Q4: Describe a plugin in webpack.

A4: A plugin in webpack is a JavaScript object that can perform a wide range of tasks during the bundling process. Plugins are used to customize and extend webpack's behavior, such as optimizing bundles, injecting variables, and more.

Q5: What is a dependency graph, and how does webpack build it?

A5: A dependency graph in webpack is a representation of the relationships between different modules or files in a project. Webpack builds the dependency graph by tracing the dependencies of the entry point and organizing them into a tree-like structure.

Q6: Webpack gives us a dependency graph. What does it mean?

A6: The dependency graph in webpack represents the relationships between different modules in a project, showing how they depend on each other. This graph is crucial for webpack to efficiently bundle and manage the project's assets.

Q7: Name some benefits of using Webpack.

A7: Benefits of using webpack include code splitting, bundling assets for efficient loading, a rich plugin system, hot module replacement for faster development, and a powerful and configurable build process.

Q8: Describe webpack runtime and manifest.

A8: The webpack runtime is a small piece of code that manages the loading and execution of the bundled modules. The manifest is a JSON file that contains the mapping between module identifiers and their corresponding output files.

Q9: List some pitfalls of Webpack.

A9: Common pitfalls in webpack include complex configuration, longer build times for large projects, steep learning curve, and occasional issues with compatibility when upgrading webpack versions.

Q10: Is it possible to use some other languages (except JavaScript) for the webpack config file?

A10: Yes, webpack supports configuration files written in JavaScript, JSON, or YAML.

Q11: Describe the tree-shaking mechanism in webpack.

A11: Tree shaking is a feature in webpack that eliminates dead code (unreferenced exports) from the final bundle, resulting in smaller and more efficient production builds.

Q12: Describe long-term caching and how to achieve this using webpack.

A12: Long-term caching involves assigning unique hashes to file names to prevent browser caching issues. Webpack achieves this by using the chunkhash in the output filename, which changes only when the content of the chunk changes.

Q13: List out some core concepts of webpack.

A13: Core concepts of webpack include entry points, output, loaders, plugins, mode, and the dependency graph.

Q14: What is the default value of entry point in webpack?

A14: The default entry point in webpack is "./src/index.js."

Q15: How can you specify different (or multiple) entry points by setting an entry property in the webpack configuration file?

A15: You can specify different entry points by configuring the entry property in the webpack configuration file as an object where keys are entry point names and values are the corresponding entry file paths.

Q16: What does the Webpack output property tell, and how can you configure this part of the process by specifying an output field in your configuration?

A16: The output property in webpack configuration specifies the output file settings. It tells webpack where to emit the bundles and how to name them. You can configure it by setting properties like filename and path.

Q17: At a high level, loaders have two properties in your webpack configuration. What are they, and how to set them?

A17: Loaders in webpack have two properties: test (specifying which files to transform) and use (specifying the loader to use for transformation). They are set as an object within the module property in the webpack configuration.

Q18: Write some differences between Loaders and plugins.

A18: Loaders are used to transform source code files, while plugins are used to perform a wide range of tasks on the entire bundle. Loaders operate at the individual file level, whereas plugins work at the bundle level.

Q19: How can you enable webpack's built-in optimizations that correspond to each environment?

A19: You can set the mode property in the webpack configuration to either "development" or "production" to enable webpack's built-in optimizations for the respective environment.

Q20: Tell the steps of webpack in processing your application.

A20: The steps in webpack processing include identifying the entry point, building the dependency graph, applying loaders to transform files, resolving dependencies, generating bundles, and optimizing output for production.

Q21: What is ESLint?

A21: ESLint is a static code analysis tool for identifying and fixing problems in JavaScript code. It helps enforce coding styles and best practices.

Q22: How is ESLint completely pluggable, and what are the commands needed to install and configure ESLint?

A22: ESLint is pluggable through the use of plugins and custom configurations. To install and configure ESLint, you can use the following commands:

bash

Copy code

npm install eslint --save-dev

npx eslint --init

Q23: Discuss the two primary ways to configure ESLint.

A23: ESLint can be configured using either a configuration file (e.g., .eslintrc.js) or inline comments within the JavaScript files.

Q24: Discuss some of the options that you can configure in ESLint.

A24: ESLint configuration options include specifying rules, defining environments, setting parser options, configuring plugins, and adjusting various other options related to code analysis.

Q25: Discuss the Different Configuration File Formats that ESLint supports.

A25: ESLint supports configuration file formats like JavaScript (.js), JSON (.json), and YAML (.yaml or .yml).

Q26: What are two ways to use configuration files? Discuss.

A26: Configuration files in ESLint can be used either globally, applying rules to an entire project, or locally, with specific configurations for individual directories or files.

Q27: Discuss the role of “Rules” in ESLint.

A27: Rules in ESLint define the patterns that should be checked in the code. They can enforce coding standards, catch common programming errors, and improve code quality.

Q28: What is an ESLint plugin?

A28: An ESLint plugin is a package that contains a set of rules, configurations, and sometimes parsers to extend ESLint's functionality for specific technologies or frameworks.

Q29: What is an ESLint parser?

A29: An ESLint parser is responsible for analyzing the code and converting it into an abstract syntax tree (AST), which ESLint then uses for static code analysis.

Q30: Differentiate between ESLint Built-in and Custom parsers with proper examples.

A30: ESLint built-in parsers are included with ESLint, like "espree." Custom parsers are external parsers that need to be installed separately. Example of custom parser: "babel-eslint."

Q31: **Explain the “array-callback-return” Rule of ESLint with proper example code.** A31: The "array-callback-return" rule enforces that array methods like **map**, **filter**, and **reduce** have a **return** statement. Example:

const result = array.map(item => item \* 2); // Error: Missing return statement

Q32: **Explain the “constructor-super” Rule of ESLint with proper example code.** A32: The "constructor-super" rule ensures that classes with a constructor must call **super()**. Example:

class MyClass extends AnotherClass {

constructor() {} // Error: 'super()' is required

}

Q33: **Explain the “for-direction” Rule of ESLint with proper example code.** A33: The "for-direction" rule checks for potentially confusing loop conditions. Example:

for (let i = 0; i < array.length; i--) {} // Error: The loop will never exit

Q34: **Explain the “no-constructor-return” Rule of ESLint with proper example code.**

A34: The "no-constructor-return" rule prohibits returning values from the constructor. Example:

class MyClass {

constructor() {

return 42; // Error: Returning values from constructor is not allowed

}

}

Q35: **Explain the “no-constant-condition” Rule of ESLint with proper example code.**

A35: The "no-constant-condition" rule flags the usage of constant conditions in if statements. Example:

if (true) {} // Error: Constant condition in if statement

Q36: **Explain the “no-dupe-args” Rule of ESLint with proper example code.**

A36: The "no-dupe-args" rule prevents duplicate function argument names. Example:

function myFunction(arg1, arg1) {} // Error: Duplicate parameter name 'arg1'

Q37: Explain the “no-dupe-class-members” Rule of ESLint with proper example code.

A37: The "no-dupe-class-members" rule identifies duplicate class member names. Example:

class MyClass {

method() {}

method() {} // Error: Duplicate class method 'method'

}

Q38: List out and discuss the “Core Rule Guidelines” of ESLint.

A38: Core rule guidelines include "Possible Errors," "Best Practices," "Strict Mode," "Variables," "Stylistic Issues," and "ECMAScript 6."

Q39: After running npm init @eslint/config, you’ll have an .eslintrc.{js,yml,json} file in your directory. Discuss "ESLint rules configured" in the above file.

A39: The "ESLint rules configured" section in the configuration file (.eslintrc.js, .eslintrc.yml, or .eslintrc.json) is where you specify the rules and their configurations for your project.

Q40: Discuss some Limitations of ESLint.

A40: ESLint has limitations, including occasional false positives or negatives, potential performance impact on large codebases, and the need for careful configuration to match project-specific needs. Additionally, it may not cover all aspects of code quality.